Pedro Miguel Simões Miranda

Licenciado em Engenharia Informática

Enabling and Sharing Storage Space Under a Federated Cloud Environment

Dissertação para obtenção do Grau de Mestre em
Engenharia Informática

Orientador : Paulo Orlando Reis Afonso Lopes, Professor Auxiliar, Universidade Nova de Lisboa

Co-orientador : Jorge Humberto Lúcio Oliveira Gomes, Investigador Principal, Laboratório de Instrumentação e Física Experimental de Partículas

Júri:

Presidente: Prof. Jácome Miguel Costa da Cunha

Arguente: Prof. Pedro Dinis Loureiro Salgueiro

Vogal: Prod. Paulo Orlando Reis Afonso Lopes

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Dezembro, 2015

Enabling and Sharing Storage Space Under a Federated Cloud Environment

Dedico este trabalho a um grande homem, com um grande coração, Humberto Miranda, que me transmitiu os princípios pelos quais me sigo hoje, e à sua mulher, a minha avó, Florinda Miranda, que tomava conta de mim quando era pequeno. Dedico este trabalho também ao meu avô Aires Simões, por todo o amor que me deu e por me ensinar que a família é tudo. E por fim, quero honrar a memória da minha avó Deolinda Simões, gostava de te ter conhecido …

Agradecimentos

A realização deste trabalho contou com muitos apoios, sem os quais a conclusão do mesmo não seria possível.

Em primeiro lugar, quero agradecer ao Prof. Paulo Lopes, por toda a paciência, apoio e disponibilidade prestado ao longo da realização desta dissertação.

Quero também agradecer ao investigador Jorge Gomes, por me receber no LIP, e guiar durante a elaboração da dissertação.

De seguida, ao Departamento de Informática da Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa, por todos os recursos materiais fornecidos ao longo do curso, e pelos professores competentes que nos permitem crescer como engenheiros.

Quero agradecer também ao Laboratório de Instrumentação e Física Experimental de Partículas, pelas excelentes condições de trabalho que me foram proporcionadas para o desenvolvimento deste trabalho.

Agradeço à equipa do grupo de computação do LIP, graças à sua experiência permitiu despistar muitos problemas com que me deparei, Mário David, João Martins, João Pina e Nuno Dias, obrigado.

Muito obrigado aos meus amigos Nuno Delgado, Nuno Correia, Filipa Lourenço e Marco Pinto, que me apoiaram não só a nível profissional como a nível pessoal. E apesar de não mencionar nomes, quero agradecer a todos os meus amigos de longa data, obrigado pela vossa amizade.

Por último, quero agradecer a toda a minha família, em especial à minha mãe, à minha irmã, aos meus tios e primos, que me apoiaram e incentivaram ao longo deste período.

Resumo

O LIP, Laboratório de Instrumentação e Física Experimental de Partículas, opera atualmente, no âmbito da Infraestrutura Nacional de Computação Distribuída (INCD), uma infraestrutura de computação de elevado desempenho (HPC) para apoio à comunidade científica nacional. Neste momento, o LIP está a desenvolver, no âmbito da referida infraestrutura, um novo serviço, este agora de computação em nuvem, cuja disponibilização pressupõe a orquestração de três sub-serviços fundamentais: computação, rede, e armazenamento.

A presente dissertação, tem por objetivo a seleção de uma arquitetura de armazenamento para um ambiente de computação em nuvem, suportando tolerância a faltas, mantendo sempre uma boa performance, e escalando facilmente face à quantidade de dados que é armazenada em ambientes típicos de HPC, enquanto corre em hardware comum. A arquitectura é focada para a sua aplicação e integração com a suite OpenStack, solução escolhida pelo LIP para o serviço de computação na cloud a disponibilizar na INCD.

Para atingir um tal objetivo, necessitamos de: a) estudar o OpenStack – a sua arquitetura e a forma como disponibiliza Infrastructure-as-a-Service (IaaS); b) fazer um estudo mais aprofundado das soluções de armazenamento passíveis de integrar no OpenStack, que precisa de armazenar: templates (imagens, na terminologia do OpenStack) de máquinas virtuais (VMs) e imagens de ISOs (CDs/DVDs), discos virtuais efémeros e persistentes para instâncias de VMs, etc.; c) apresentar um estudo preliminar de três sistemas de ficheiros que são sérios candidatos para integrar no OpenStack: NFS, Ceph e GlusterFS; e, d) escolha de um sistema candidato a integrar com o OpenStack, efectuando uma avaliação experimental do sistema escolhido.

Palavras-chave: Armazenamento em nuvem, computação em nuvem, virtualização, Sistemas de armazenamento de objetos, sistemas de armazenamento de ficheiros.

Abstract

To support the Portuguese scientific community LIP, Laboratório de Instrumentação e Física Experimental de Partículas, has been operating a high performance computing (HPC) infrastructure under a grant from the Infraestrutura Nacional de Computação Distribuída (INCD) Program. Now, LIP is promoting another initiative towards the same community, that is, to build a Cloud Computing (CC) service which orchestrates the three fundamental resources: compute, network, and storage.

The main goal of this dissertation is to research, implement, benchmark and adopt the most appropriate backend storage architecture for the OpenStack cloud computing software service, chosen by LIP (following EGI, the European Grid Infrastructure) to be the cloud platform to be deployed in the new CC-INCD program.

For this work, our objectives are: a) to gain an understanding of OpenStack – its architecture, and the way it works to offer an Infrastructure-as-a-Service (IaaS) platform; b) look for candidates suitable to be deployed as OpenStack' storage backends, which should be able to store templates (images, in the OpenStack terminology) of virtual machines (VMs) and ISO images (CDs/DVDs), ephemeral and persistent virtual disks for VM instances, etc.; c) to present a preliminary study of three file systems that are strong candidates to be integrated with OpenStack: NFS, Ceph and GlusterFS; and, d) to choose a candidate to integrate with OpenStack, and perform an experimental evaluation.

Keywords: Cloud Storage, Cloud Computing, Virtualization, File Systems.

Conteúdo

Lista de Figuras

Lista de Tabelas

1

Introdução

1.1 Motivação

O Laboratório de Instrumentação e Física de Partículas (LIP) tem vindo a operar uma infraestrutura de computação de alto desempenho, através de um programa da Infraestrutura Nacional de Computação Distribuída (INCD), de forma a suportar algumas necessidades da comunidade científica portuguesa. Do ponto de vista do armazenamento esta infraestrutura, com um espaço útil próximo de 1 PB, utiliza o Lustre [BZ02], um sistema de ficheiros (SF) distribuído capaz de suportar as necessidades de desempenho de aplicações sequenciais (mercê de uma API POSIX) e paralelas (para as quais disponibiliza uma interface MPI-IO); o SF Lustre é suportado, ao nível físico, por mecanismos de resiliência, tais como volumes RAID-5.

1.2 Definição do problema

No âmbito do supramencionado programa INCD o LIP está atualmente a disponibilizar uma infraestrutura OpenStack, que segue o modelo de serviço IaaS, de computação em nuvem como alternativa à infraestrutura de alto desempenho. As razões para tal são variadas, e vão desde a possibilidade de aumentar a satisfação dos utilizadores por conseguir diminuir o tempo de espera para entrada em execução dos seus trabalhos (e melhorar a utilização dos recursos) até à possibilidade de oferecer aos utilizadores a oportunidade de usarem as suas próprias distribuições, configurações e *stacks* de software, com privilégios de administração.

Contudo, o Lustre não é o sistema de ficheiros mais indicado para suportar uma tal infraestrutura, pelo que, neste trabalho, se procuram encontrar e avaliar outros mais adequados, a ser integrados com o OpenStack e, porventura, possam vir no futuro a substituir o Lustre, reduzindo-se assim o leque de SF a manter no parque do LIP.

1.3 Objectivos

O trabalho que realizamos pode, grosso modo, ser subdividido em duas fases, sendo a primeira realizada durante a Unidade Curricular de "Preparação de Dissertação" e a segunda no período de Elaboração de Dissertação. Assim, na primeira fase definiram-se, e atingiram-se, os seguintes objectivos: a) o estudo do paradigma *cloud*, em geral, e o de uma *cloud* OpenStack, que disponibiliza Infraestrura(s) como Serviço (IaaS), em particular; b) investigar sistemas de armazenamento candidatos a serem integrados com os serviços de armazenamento OpenStack. Esta fase culminou com o estudo preliminar de quatro sistemas identificados como fortes candidatos a integrar com o OpenStack – NFS, GlusterFS, Sheepdog e Ceph – para os quais vertemos, nesta dissertação, as conclusões a que chegamos. O estudo destes sistemas específicos é fruto de uma pesquisa realizada, e deve-se ao facto de haver uma forte aderência na integração destes sistemas de armazenamento com o OpenStack e ao consequente desenvolvimento que permite introduzir novas funcionalidades. O Ceph é um requisito imposto inicialmente, dado que um dos parceiros do projecto (o CERN) tomou a decisão de usar o Ceph.

Na segunda fase, o nosso objetivo inicial era o de realizar a instalação dos quatro sistemas de armazenamento (SA) candidatos e proceder à sua avaliação; tal não foi contudo possível por duas razões: em primeiro lugar, a infraestrutura de testes é grande e dispendiosa, pelo que se procurou minimizar o tempo durante o qual estaria a ser utilizada exclusivamente para a instalação e testes de desempenho; em segundo, tal como já foi afirmado, porque um dos parceiros do projecto (o CERN) tomou a decisão de usar o Ceph, o que de certa forma coartou a possibilidade de outros parceiros usarem outro SF, já que se pretende federar as diversas clouds. Houve também necessidade de colocar a infraestrutura OpenStack em produção, não sendo depois possível avaliar o Ceph nesse ambiente, pelos prejuízos que os ensaios poderiam trazer aos utilizadores; optou-se, por

isso, por realizar a avaliação usando apenas a camada de virtualização, KVM, usada na infraestrutura Ceph – o que não invalida as conclusões pois é essa camada que exercita o *startup/shutdown* e as E/S das VMs.

Assim, e resumindo, os objetivos a que nos propomos no presente trabalho são: a) Estudar, analisar e comparar um conjunto de sistemas de ficheiros capazes de serem integrados no OpenStack e que suportem as suas diversas necessidades de armazenamento - imagens, objetos de grandes dimensões, podendo ser *templates* e instâncias de máquinas virtuais (VMs); e também volumes para fornecer armazenamento baseado em blocos a VMs. b) Avaliar o desempenho bruto (raw) individual dos subsistemas principais da infaestrutura de armazenamento disponibilizada para esta dissertação: rede (*Ethernet* 10 Gbps), armários de discos, HBAs (*Host Bus Adapters*) – dispositivos de interligação armário/servidor. c) Avaliar escalabilidade da infraestrutura de armazenamento em função do número de discos/armário, do número de armários/servidor, e da utilização de *striping* (RAID-0) versus *striping* com tolerância a faltas (RAID-5); d) Avaliar o desempenho das melhores configurações obtidas em (c) usando sistemas de ficheiros locais, e usando o Ceph; e) Avaliar o desempenho da melhor configuração obtida em (d) quando integrada num subsistema de virtualização de servidores.

1.4 Contribuições

A contribuição principal desta dissertação é, naturalmente, a análise do comportamento e desempenho do Ceph, tanto na perspectiva de um sistema de armazenamento "object storage" per si, como quando usado como "backend" de um hipervisor (e, consequentemente, do desempenho expectável quando utilizado com o OpenStack). Como contribuições secundárias, a mais importante é certamente a comparação dos desempenhos de uma solução com tolerância a faltas baseada exclusivamente nos mecanismos de replicação do Ceph, i.e., em software, versus uma solução que usa exclusivamente hardware/-firmware para atingir o mesmo objetivo. Como menores contribuições, podemos referir a análise ao desempenho da infraestrutura de armazenamento disponibilizada, bem como escolha das ferramentas e definição de uma metodologia que, aplicada de forma sistemática, pode ser usada para outras avaliações e permitir comparações entre infraestruturas e sistemas de armazenamento, hardware e software, distintos.

1.5 Organização do documento

O resto do presente documento está organizado da seguinte forma: na secção 2 apresentam-se os conceitos fundamentais e o estado da arte das tecnologias que estão na base do trabalho realizado, começando pela virtualização (de servidores, de rede e de armazenamento) e pela computação em nuvem (incluindo os modelos de implantação e de serviço mais comuns); segue-se, ainda na secção 2, uma descrição sucinta mas completa da arquitetura do OpenStack, a solução de *cloud* IaaS em funcionamento no LIP. Na secção 3,

discutem-se em detalhe os componentes e serviços OpenStack que interagem com os subsistemas de armazenamento, e a forma como a arquitetura permite a substituição e/ou integração com subsistemas externos ao OpenStack - no caso NFS, Ceph, GlusterFS e Sheepdog. Na secção 4 descreve-se a avaliação: a metodologia, a avaliação dos aspectos de infraestrutura (rede, discos, armários e servidores e seus HBAs), de escalabilidade desta (múltiplos discos por armário e múltiplos armários por servidor), a influência dos mecanismos de tolerância a faltas por hardware/firmware no desempenho do subsistema de armazenamento, o desempenho e escalabilidade do Ceph e, por fim, o desempenho deste quando integrado no hipervisor KVM. A dissertação termina com a secção 5, na qual se descrevem as conclusões e se deixam pistas para trabalho futuro.

2

Estado da Arte

Este capítulo apresenta o estado da arte em tecnologias de computação em nuvem, seus modelos de serviço e de implantação, e são apresentadas algumas soluções adotadas aquando da utilização de *clouds* para a execução de aplicações de computação científica, nomeadamente ao nível do armazenamento. Em particular, na secção 2.1, é introduzido o conceito de virtualização, fundamental para desacoplar os recursos dos dispositivos físicos que os suportam; na secção 2.2, é descrito o sistema de virtualização KVM; na secção 2.3 é apresentado o conceito de computação em nuvem bem como os modelos mais comuns de serviço e implantação. Finalmente, na secção 2.4 descrevemos resumidamente a arquitetura do OpenStack, uma plataforma de computação em *cloud* em código aberto.

2.1 Virtualização

Em Ciência da Computação, o termo virtualização é utilizado para definir algo que parece existir e, no entanto, não é real; a sua utilização é ampla, cobrindo várias subáreas da Ciência da Computação, como por exemplo o conceito de memória virtual de um sistema de operação, ou o conceito de ecrã virtual de um computador.

Nesta dissertação, o termo virtualização é usado no contexto estrito daquilo que se convencionou designar por virtualização de servidores e que engloba um conjunto de tecnologias que permite a criação de servidores virtuais (incluindo CPU, memória e dispositivos de entrada e saída) cuja arquitetura segue aproximadamente a arquitetura do servidor físico, mas inclui também aspetos de virtualização de redes e de armazenamento.

2.1.1 Evolução e Tipos de Virtualização

A história da virtualização começou nos finais da década de 60, com a IBM, que decide particionar logicamente o *mainframe* de computadores em várias máquinas virtuais, permitindo a sua execução "simultânea" sob a forma de tempo-partilhado (*time-sharing*); este solução permitiu a partilha de recursos muito dispendiosos por um grupo alargado de utilizadores, aumentando a eficiência no uso dos mesmos e diminuindo os custos associados.

Nos meados dos anos 80 a 90, a indústria migrou de arquiteturas centralizadas baseadas num só mainframe para as distribuídas, usando múltiplos servidores x86, mais simples e baratos, e suficientes para as necessidades das organizações/PMEs e, por isso, a evolução da virtualização estagnou um pouco. Apenas nos finais dos anos 90, a VMWare introduziu a virtualização para sistemas x86, o que abriu de novo um leque de para aplicações para esta tecnologia.

Atualmente, a virtualização continua a ser uma solução robusta que as organizações têm ao seu dispor para utilizar eficientemente recursos computacionais. Nos centros de dados utilizam-se técnicas de virtualização para abstrair o *hardware* dos servidores, criando repositórios de recursos lógicos tais como CPUs, memória, discos e rede, entre outros, recursos que podem depois ser agrupados para formar as "infraestruturas" sobre as quais se vão executar serviços e aplicações de uma forma muito mais eficiente e económica.

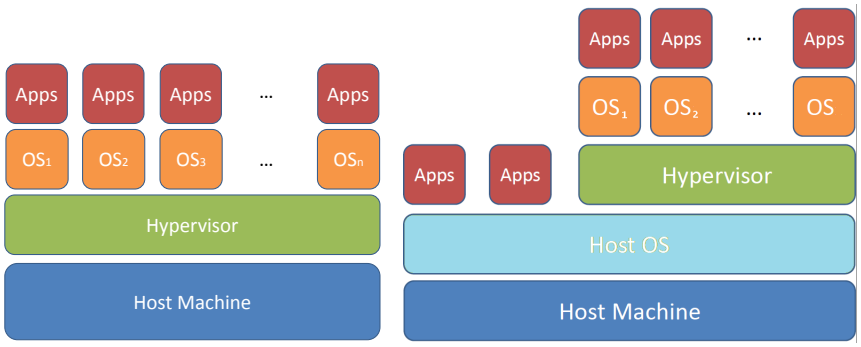
Assim, da mesma forma que nos é conveniente abordar separadamente os três tipos de recursos que compõem uma infraestrutura computacional – servidores, rede e dispositivos de armazenamento – também é conveniente abordar separadamente a virtualização de cada um.

2.1.2 Virtualização de Servidores

Atualmente, a virtualização de servidores, é o tipo mais comum de virtualização; neste caso, o *software* de virtualização executa no servidor físico e consolida múltiplos servidores virtuais e todos os seus recursos (RAM, CPU, etc); isto tem várias vantagens, tais como reduzir os custos administrativos, simplificação no *upgrade* da infraestrutura existente, simplificação do planeamento de recuperação de catástrofes, possibilitar ainda utilizar aplicações que são incompatíveis com os sistemas operativos existentes nas máquinas ao criar de uma forma muito simples uma máquina virtual com o sistema operativo compatível com a aplicação, isto também se aplica caso haja uma necessidade de testar a aplicação em ambientes de teste em máquinas virtuais que podem ser facilmente eliminadas.

A camada de virtualização, cria as abstrações dos recursos (vCPUs, vRAM, etc.) que são depois usadas para constituir as máquinas virtuais (VMs); para tal camada, existem duas implementações: a de tipo I e a de tipo II (Figura 2.1). Nas arquiteturas de

virtualização de tipo I, também chamadas nativas (ou *bare metal*), a camada de virtualização é executada diretamente sobre o *hardware*; alguns exemplos de tipo I são: Citrix Xen Server, VMware ESXi, Oracle VM Server, Kernel Virtual Machine (KVM), e Microsoft Hyper-V. Nas arquiteturas de tipo II o hipervisor (nome pelo qual é conhecida a camada de virtualização) é executado como um processo num SO, fazendo uso dos seus serviços, nomeadamente no acesso a dispositivos de E/S; são exemplos de hipervisores de tipo II o VMware Workstation, Player e Fusion e o Oracle VirtualBox.



(a) Hipervisor tipo I. (b) Hipervisor tipo II.

Figura 2.1: Arquiteturas de hipervisores tipo I e II.

Uma arquitetura *hardware* (e correspondente hipervisor) dizem-se capazes de oferecer virtualização total (*full virtualization*) quando os recursos abstratos que este implementa são indistinguíveis dos reais; tal apenas é possível se a arquitetura, e em particular o CPU, obedecem a determinados princípios de desenho, como era o caso dos CPUs IBM 360 e seguintes. Nesses casos, um qualquer sistema hospedado (*guest*) pode ser executado, sem modificações, na VM. Quando tais princípios não são observados, há duas soluções: ou se reescreve o SO de forma a não usar as "instruções problemáticas" (*sensitive instructions*), numa técnica conhecida como para-virtualização, ou se interpreta cada instrução oriunda do *guest*, intercetando e reescrevendo as que são problemáticas, numa técnica conhecida como tradução binária.

No caso das diferentes encarnações da arquitetura x86 Intel/AMD até cerca de 2005 [Vmw], a única forma de aproximar uma virtualização completa (*full*) era recorrer à técnica da tradução binária, que foi usada pela VMware com muito sucesso, embora o desempenho fosse, em alguns casos, bastante penalizado quando comparado com a para-virtualização aplicada pela Citrix aos sistemas Linux. A primeira arquitetura x86 com suporte para virtualização assistida pelo *hardware* aparece então em 2006, com a inclusão de um conjunto adicional de um anel de proteção (privilégios de execução) desenhado para executar o hipervisor enquanto os *guests* continuavam a correr nos anéis 0 em *kernel mode*, e 3 em *user mode*.

Tradução binária

Na interpretação e eventual tradução (Figura 2.2), todas as instruções oriundas do *guest* são intercetadas pelo hipervisor. No caso das instruções não-sensíveis (a maioria das não-privilegiadas), o hipervisor promove a sua execução direta no *hardware*; no caso das privilegiadas, o "trap" gerado pelo CPU é tratado pelo hipervisor, que faz a emulação da interrupção (i.e., produz no *guest* os mesmos efeitos da execução da instrução); finalmente, no caso das instruções "problemáticas" o hipervisor reescreve dinamicamente o código oriundo do CPU por código equivalente extirpado das referidas instruções.

Note-se que, nesta técnica, o *guest* SO é carregado no anel 1, e não no 0 como seria numa execução nativa desse SO; já o hipervisor é carregado no anel 0, o que significa que "apanha" naturalmente todas as interrupções *hardware* e *software* (*traps*).

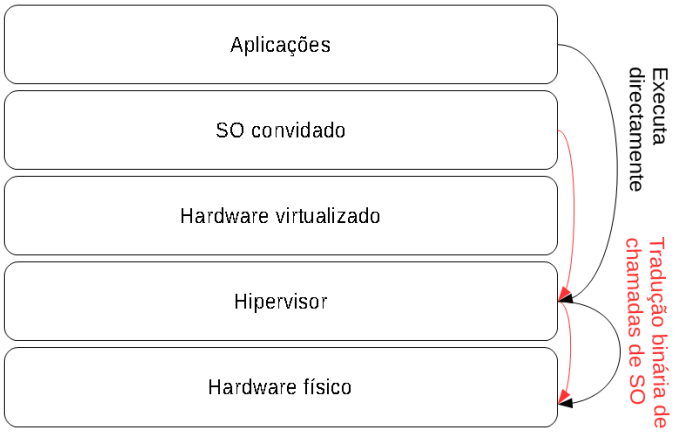


Figura 2.2: Tradução binária.

Para-virtualização

A para-virtualização (figura 2.3) é, como dissemos anteriormente, uma técnica de virtualização, que obriga à modificação do *kernel* do sistema hospedado para o extirpar de zonas de código contendo instruções sensíveis, e substituindo-as por chamadas ao hipervisor (*hypercalls*), chamadas essas desencadeadas por um mecanismo de software *interrupts* (*traps*) tradicional. Esta solução obtém um ganho significativo em relação à tradução binária nos casos em que as aplicações em execução no *guest* fazem uso intensivo de serviços do *kernel* – como é, por exemplo, o caso dos sistemas de bases de dados (SGBDs). Têm contudo o inconveniente de necessitar de *kernels* modificados, o que obriga à existência de código fonte do SO disponível para tais modificações; a título de exemplo, apenas o Windows XP foi "para-virtualizado" (pela Microsoft Research e um grupo de investigação em Sistemas de Operação da Universidade de Cambridge) e somente para estudo [BDFHHHNPW03].

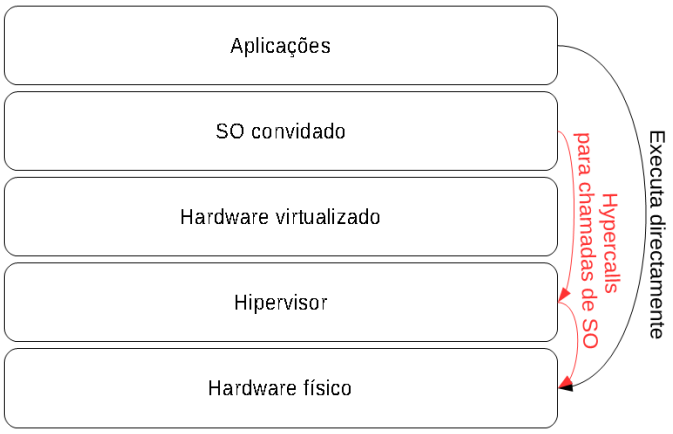


Figura 2.3: Para-virtualização.

Virtualização Assistida por Hardware

A virtualização assistida por *hardware* foi introduzida pela Intel (VT - Virtualization Technology) e AMD (SVM - Secure Virtual Machine) como forma de oferecer a virtualização completa de forma simples e eficiente, dispensando a tradução binária e a para-virtualização. Conforme se pode ver na Figura 2.4, o sistema hospedado corre de novo no seu anel de proteção natural (nível 0), tendo o hipervisor sido transferido para um novo anel, -1. Assim, agora apenas a execução das instruções privilegiadas e sensíveis se traduz em *traps* invocando o hipervisor, sendo que a generalidade das instruções é diretamente executada no CPU.

Para terminar vale a pena referir que a existência de VT/SVM não eliminou completamente a para-virtualização, assistindo-se à sua utilização na produção de *drivers* (ditos para-virtualizados) para *hardware* que só existe ao nível dos hipervisores e das VMs que nele se executam. Como exemplo, podemos referir que a VMware aconselha que nas suas VMs se usem NICs vmxnet3 em vez de emulações de verdadeiros Intel E1000; o resultado é um muito melhor desempenho com menor consumo de CPU.

2.1.3 Virtualização de Armazenamento

A virtualização de armazenamento permite desacoplar o armazenamento físico, constituído por discos (magnéticos ou outros) do armazenamento lógico, constituído por volumes sobre os quais eventualmente se "instala" um sistema de ficheiros. Estes volumes são abstrações (vectores de blocos) oferecidas por um gestor de volumes lógicos (logical volume manager) que pode ser implementado integralmente em software, como no caso do Linux LVM ou o Windows Volume Manager, ou por *hardware/firmware* em placas controladoras de disco e armários de discos (*disk arrays*). Estes gestores de volumes podem "conferir" aos volumes propriedades não existentes nas unidades de disco usadas para criar esses mesmos volumes, sendo o exemplo mais comum a oferta de volumes com

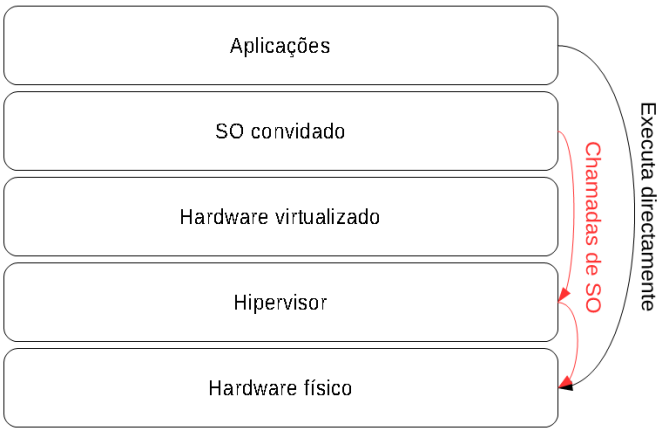


Figura 2.4: Virtualização assistida por hardware.

propriedades de tolerância a faltas por utilização de tecnologias RAID 1, 3 ou 5 ou desempenho acrescido, como é o caso do RAID 0 [Pat86]; nestes casos os diferentes volumes são referenciados através do seu LUN (Logical Unit Number).

A redundância ao nível dos discos (volumes) é um requisito base para a disponibilização de sistemas de alta disponibilidade, i.e., sistemas capazes de rapidamente recuperar de faltas e repor em funcionamento os seus serviços; mas, para que tal seja conseguido, os volumes não podem estar assentes sobre discos internos dos servidores, pois assim seria necessária intervenção humana para os colocar noutro servidor. Vulgarizam-se assim nos centros de dados (CD), a partir da década de 90 do séc XX, as redes de armazenamento, Storage Area Networks (SAN), assentes em infraestruturas Fibre Channel (FC) interligando servidores a *disk arrays*, sendo que estes últimos disponibilizam volumes de dados aos servidores (que os formatam usando um sistema de ficheiros tradicional – ext3, XFS, NTFS, etc. – ou um sistema de ficheiros especializado – GFS, GPFS – que permite o volume ser partilhado simultaneamente por vários servidores. Assim, em caso de falha de um servidor, um outro poderá rapidamente "mapear" o(s) volume(s) do servidor avariado, e aceder ao seu conteúdo, re-disponibilizando o(s) serviço(s). Já entrados no séc. XXI, a existência de duas infraestruturas distintas nos CDs, uma usando tecnologias e protocolos FC para "movimentar" blocos de dados, e outra usando tecnologia Ethernet e TCP/IP para movimentar "outra informação" acaba por redundar numa "fusão" dos dois tipos de utilização numa única rede, baseada em Ethernet e TCP/IP, tendo sido criado um novo protocolo, iSCSI, para substituir o FC.

Contrastando com o anteriormente exposto, uma outra forma de promover arquiteturas tolerantes a faltas assenta na utilização de servidores de ficheiros como forma de partilhar dados que se mantêm externos aos sistemas que os utilizam; são bem conhecidos os sistemas de ficheiros de tipo cliente-servidor NFS e CIFS (por vezes designado SMB ou Samba). Este paradigma, conhecido como Network-Attached Storage (NAS) contrasta,

como dissemos, com o anterior porque i) os "volumes" (árvores ou sub-árvores de ficheiros) exportados são por natureza partilhados, e ii) na rede que interliga clientes (os servidores que consomem os ficheiros) a servidores (os *disk arrays*, agora com interfaces Ethernet em vez de FC) movimentam-se "partes" de ficheiros e metadados, em vez de blocos.

Mais recentemente, um outro paradigma de armazenamento ganha especial relevância face à sua escalabilidade intrínseca – em termos de capacidade bruta de armazenamento e de desempenho – e ao seu potencial para ser uma solução mais económica que os *disk arrays* tradicionais; referimo-nos ao Object Storage (OS), uma tecnologia baseada na utilização de nós de armazenamento que são servidores comuns (dotados de discos internos) agrupados em "*pools*" que oferecem volumes tolerantes a faltas aos servidores; ao contrário de um sistema de ficheiros (SF) que contém os dados contidos numa estrutura hierárquica (árvore), num OS os dados são armazenados em objetos numa pool (num estrutura "*flat*"), em que para aceder a um objeto é necessário fazer o pedido pela uma chave, o que em troca retorna um valor, ou seja, os dados.

Num OS, a rede que interliga os nós de armazenamento e os servidores "consumidores" é usada não apenas para movimentar blocos entre os servidores e os volumes "localizados" nos nós de armazenamento, mas para manter entre estes réplicas, distribuídas ou não, dos referidos volumes. Sendo um volume um vector de *bytes*, estes sistemas mapeiam os volumes em ficheiros; são, por isso, dotados internamente de um SF, sendo que estes podem ser muito simples, pois o seu espaço de nomes é plano (*flat file system*). Contudo, a maior parte dos sistemas de armazenamento do tipo OS existentes (e.g. Ceph) utiliza um SF tradicional, por exemplo, o XFS. Assim, as operações sobre volumes (criar, apagar, duplicar/clonar) não são mais que simples operações sobre ficheiros (muitas vezes muito grandes, da ordem dos TB). Um aspeto a destacar nos sistemas de armazenamento deste tipo é a sua API: ainda que podendo disponibilizar uma API mais tradicional (e.g., iSCSI), favorecem em especial as APIs proprietárias baseadas em HTTP/REST.

Qualquer das infraestruturas acima descritas, SAN, NAS ou OS pode ser usada para suportar as necessidades de armazenamento de um hipervisor, que precisa de "locais" onde guardar os "dados" necessários para suportar a meta-informação relativa às VMs, e os conteúdos das próprias VMs.

2.1.4 Virtualização de Rede

De uma forma análoga ao anteriormente descrito a propósito da virtualização do armazenamento, a virtualização de rede desacopla entidades lógicas – redes, switches, placas de rede (*network interface cards*, NICs) – dos respectivos dispositivos ou entidades. Assim podemos, numa infraestrutura virtualizada existente num servidor dotado de um hipervisor, como na representada na Figura 2.5, criar VMs e dotá-las de vNICs (NICs virtuais),

ligar estes a vSwitches (switches virtuais) e associar estes últimos aos pNICs (NICs físicos) que se encarregarão de fazer fluir o tráfego entre as VMs e o "exterior" (sendo que o tráfego entre VMs em execução no mesmo hipervisor muito provavelmente fluirá na rede virtual sem necessidade de passar pelo exterior, a menos que um protocolo de *routing* a isso obrigue). Note-se que, tal como os pNICs, os vNICs têm endereços MAC que permitem que uma VM seja visível externamente e, do ponto de vista da rede, indistinguível de um host físico.

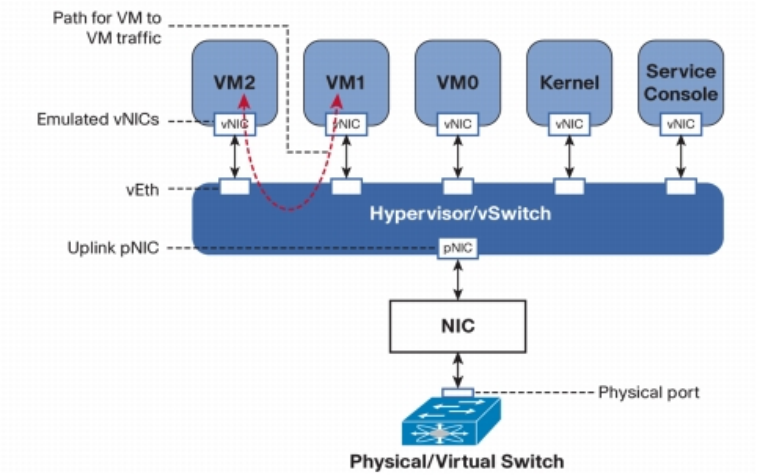


Figura 2.5: Exemplo de infraestrutura de rede com virtualização.

2.1.5 Virtualização: para além do hipervisor

Uma infraestrutura de virtualização precisa, para ser útil e prática, de oferecer funcionalidades que ficam para além das oferecidas pelo hipervisor. Referimo-nos a aspetos fundamentais da gestão de uma infraestrutura: executar, suspender e terminar a execução de uma VM são obviamente funções básicas do hipervisor, assim como a capacidade de salvaguardar o estado (uma operação designada *snapshot*), num dado instante, dos recursos da VM para posteriormente se regressar a esse mesmo estado, se necessário. Mas é também preciso ser capaz de criar e destruir as VMs, poder copiá-las (ou cloná-las, que é o termo usado neste caso), para efeitos de criação de novas VMs ou como mecanismo de cópia de salvaguarda (*backup*). Alguns produtos de virtualização, como é o caso do VMware vSphere oferecem ainda funcionalidades muito mais avançadas, como de criação de thin clones (baseados em *snapshots* efectuados sobre *templates* e que evoluem a partir destes usando mecanismos *copy-on-write*), ou a possibilidade de definir clusters de servidores de virtualização, cada um executando a sua cópia local do hipervisor, mas dotados de um sistema de escalonamento de recursos que a cada instante pode decidir qual o servidor mais conveniente para executar uma VM, e migrar VMs de um nó para outro para melhor distribuição de carga, para efectuar operações de manutenção em nós, etc..

2.2 Estudo de Caso: Kernel Virtual Machine (KVM)

O KVM, o hipervisor utilizado neste trabalho, é um sistema de virtualização que utiliza suporte de virtualização via hardware, estando disponível como modulo do *kernel* Linux, estando incluído no *kernel* a partir da versão 2.6.20. O KVM foi desenhado para tirar partido de vários mecanismos utilizados pelo sistema operativo e também para utilizar o suporte hardware para virtualização discutidos anteriormente.

Num ambiente Linux, cada processo executa em modo utilizador ou kernel; no entanto, o KVM introduz outro conceito, o modo convidado (ou *guest-mode*), com o objectivo de executar operações do SO da VM com privilégios menos estritos (não sendo possível executar em nível 0).

O desenvolvimento do KVM teve em conta a reutilização dos recursos de gestão do Linux. A gestão de memória sofreu modificações com a introdução do KVM; foi adicionado shadow page tables pois não existia suporte *hardware* para a gestão de memória, mais tarde e como já foi dito, a Intel e a AMD implementaram este suporte. O escalonador de um sistema operativo fornece tempo de CPU para cada processo poder executar as suas tarefas, implementa filas e organiza as tarefas com estratégias próprias, dado isto, e como o objetivo do KVM foi reutilizar os mecanismos já existentes, as VMs foram implementadas como processos, e obtêm o tempo de CPU em conjunto como qualquer outro processo.

A interface de controlo KVM fica visível (em `/dev/kvm`) quando o módulo KVM é carregado, e permite a comunicação com o hipervisor através de um conjunto de chamadas de sistema ioctls [KKLLL07]; estas chamadas são por isso, utilizadas para definir operações do hipervisor tais como a criação de VMs, alocação de memória para uma VM, alocar e executar CPUs virtuais.

Tendo em conta o que foi discutido até aqui, faz sentido falarmos da virtualização dos dispositivos E/S, o mesmo é realizado em modo utilizador através de uma versão modificada do QEMU. Para cada VM é criado um processo QEMU que simula os vários dispositivos E/S da VM. Quando uma VM executa uma operação de E/S, o KVM interceta e redireciona para o processo QEMU correspondente.

O modelo de execução do KVM (ver a Figura 2.6, retirado de [AK07]) apresenta o fluxo definido entre os vários componentes deste sistema de virtualização.

Gestão de VMs e definição do respetivo *hardware* virtual

A fim de realizar o estudo presente neste trabalho, foi necessário utilizar um hipervisor, neste caso o KVM, dado que este é muito utilizado para as instalações no ambiente de armazenamento em questão. No entanto, a instalação OpenStack, a qual também necessitará de um hipervisor, não está de maneira nenhuma acoplada ao KVM, pelo contrário, esta utiliza um *middleware* chamado libvirt, cujo objectivo principal é fornecer uma única via para de gestão de ambientes virtualizados, suporta integração com vários hipervisor,

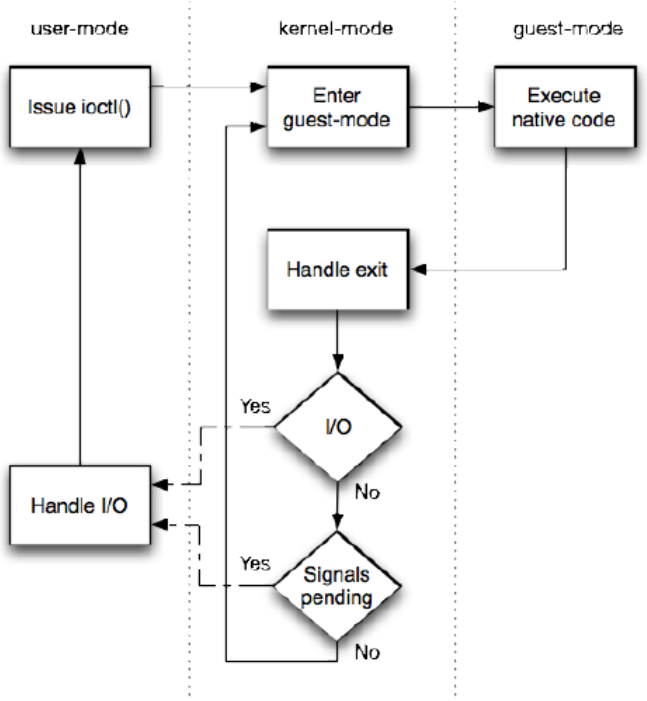


Figura 2.6: Modelo de execução do KVM

tais como KVM, Xen, VMware ESXi, VMware Workstation/Player, Hyper-V da Microsoft, entre outros.

A instalação do libvirt inclui APIs, um *daemon* (libvirtd) e também uma *command line interface* (CLI), virsh. Usualmente, a gestão de máquinas virtuais num ambiente KVM é feito recorrendo a uma *interface* de linha de comandos, ou seja, recorrendo à execução de comandos virsh, com auxilio de outras CLI tais como virt-viewer para visualização das VMs, virt-clone para operações de clonagem de VMs; ou através da interface gráfica virt-manager, estas são implementadas com a API do libvirt; estas ferramentas permitem a definição (através de XML) do *hardware* virtual dos sistemas convidados, o qual permite de uma forma prática a conceção dos recursos para a VM utilizar.

O libvirt integrado com o KVM, permite traduzir várias operações existentes no ciclo de vida de uma VM, estes podem ser executado através do virsh, os comandos são: `start`, `stop`, `pause`, `save`, `restore` e `migrate`; outros comandos virsh são necessários previamente ao ciclo de vida da VM, ou seja, são necessários para a definição da VM.

2.3 Computação em Nuvem

O recurso à virtualização originou poupanças, muito visíveis ao nível dos custos de aquisição e dos gastos de energia, e encurtou consideravelmente o tempo de espera pela entrega de servidores (e, portanto, até um certo ponto, de aplicações e serviços) que se traduzem em melhorias nos serviços de TI que suportam as empresas. No entanto, algumas

tarefas continuaram a ser bastante morosas, como a instalação e/ou (re-) configuração de software base (SO, etc.), a afetação de recursos de rede (interligar as VMs aos switches apropriados, configurar o isolamento entre redes, definir o endereçamento e o encaminhamento, etc.) e de armazenamento. E, quando tais recursos deixam de ser necessários (por exemplo, VMs disponibilizadas para desenvolvimento e/ou testes) a sua "destruição" (acompanhada das necessárias tarefas administrativas – anotar IPs livres, portas em switches, espaços em disco, etc.) é igualmente uma tarefa morosa.

Tudo estava pronto, então para a computação em nuvem, que, segundo o NIST (National Institute of Standards and Technology) se define como "... a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.".

A virtualização aparece assim como a tecnologia base que permite criar pools de recursos partilhados, e um rápido aprovisionamento e libertação dos mesmos; um sofisticado e complexo sistema de orquestração permitirá fazer essas tarefas com um mínimo de interação humana. O acesso remoto à infraestrutura quer-se ubíquo e simples, pelo que a mediação através de portais (de utilizador, de consumidor e de provedor) aparece naturalmente, e o aspeto "a pedido" reforça a noção de que, para o utilizador, os recursos só precisam de ser afetados apenas quando necessários.

2.3.1 Modelos de Implantação

Os modelos de implantação distinguem-se em termos de propriedade (*ownership*) da nuvem, ou seja: a quem pertence, quem gere, quem opera e quem consome os serviços disponibilizados pela plataforma.

Nuvem Privada É operada unicamente por uma organização, e pode ser gerida por essa mesma organização ou por uma entidade externa. Habitualmente, a organização adquire todos os recursos, incluindo o espaço necessário, hardware, software e equipamentos de energia e controlos ambientais (*Heat, Ventilation and Air Conditioning* - HVAC), o que significa que é um investimento muito dispendioso. Tem, sobre a sua congénere pública a vantagem da propriedade dos dados ser indiscutível, já que estes estão contidos no perímetro da organização.

Nuvem Pública Uma nuvem é considerada pública quando o os seus serviços são disponibilizados para utilização de um público externo (à organização que detém os recursos) que, naturalmente não necessita de dispor de infraestrutura própria. Sendo infraestruturas de grande dimensão, o seu grau de elasticidade – capacidade de afetar dinamicamente recursos a pedido e de os libertar quando não mais necessários – é geralmente muito superior ao existente em *clouds* privadas. As questões relacionadas com a propriedade, a confidencialidade, e a segurança dos dados têm sido objecto de grande debate e estudo; por exemplo, alguns provedores de *cloud* têm criado centros de dados

na Europa para satisfazer as legislações de alguns países europeus.

Nuvem Comunitária É uma nuvem partilhada entre organizações distintas, mas com objectivos semelhantes (por exemplo, um grupo de municípios); podem ser geridas internamente, por uma ou mais das organizações participantes, ou por uma entidade externa. Os custos são, em geral, divididos entre as organizações o que muitas vezes é o bastante para tornar a aquisição e operação possíveis.

Nuvem Híbrida É um modelo pelo qual uma dada nuvem, tipicamente privada, requisita temporariamente recursos a uma outra, tipicamente pública, para responder a picos de tratamento que necessitam de recursos que estão para além da capacidade da requerente. Ao contrário dos modelos anteriores, nos quais há geralmente uma relação de um-para-um entre a nuvem e a pilha de software que implementa os serviços de *cloud*, é comum nas nuvens híbridas a existência de pilhas distintas – por exemplo, uma organização pode usar o VMware vCloud na sua *cloud* privada e "fazer *burst*" (solicitar recursos adicionais) à *cloud* Amazon EC2.

2.3.2 Modelos de Serviço

A disponibilização de serviços de computação na nuvem pode ser efetuada segundo três paradigmas distintos: Infraestrutura-como-Serviço (*Infrastructure-as-a-Service*, ou IaaS), Plataforma-como-Serviço (*Platform-as-a-Service*, ou PaaS), e Software-como-Serviço (*Software-as-a-Service*, ou SaaS); no entanto, mais recentemente, têm surgido muitos mais, de tal forma que já se fala de XaaS – *Anything-as-a-Service*.

Infraestrutura-como-Serviço No modelo IaaS, o consumidor adquire[1] recursos: servidores (virtuais, quase sempre) com uma dada capacidade, expressa em número de *cores*, dimensão da memória, espaço de armazenamento em disco, largura de banda das interligações de rede, etc.. No caso da IaaS ser do tipo privado, o consumidor pode até instalar na VM o sistema de operação e *stack* de software que desejar, mas nas insfraestruturas IaaS públicas já não se passa assim: o consumidor pode escolher de entre um conjunto de imagens pré-definidas (designadas *templates*, ou *golden images*). A solução software de código aberto mais divulgada para *clouds* IaaS é sem dúvida o OpenStack; já no domínio das soluções comerciais as mais comuns são o VMware vCloud e o HP Helion. A Amazon, a Google, a IBM, a Microsoft e a Rackspace são exemplos de provedores públicos IaaS que usam, respetivamente, os stacks EC2 (Elastic Cloud 2), Google CE (Compute Engine), IBM SmartCloud Enterprise, Azure, e Rackspace Open Cloud (baseada no OpenStack).

Plataforma-como-Serviço

No modelo PaaS o fornecedor entrega uma plataforma de computação já assente numa infraestrutura; fornecendo todo o *middleware* necessário, tal como o ambiente de

[1]Note-se que o "adquire" não pressupõe um pagamento efetivo, pois o modelo IaaS tanto pode ser usado numa nuvem pública como privada.

execução da linguagem de programação, base de dados, servidores Web, etc., de forma que uma equipa de desenvolvimento de aplicações pode trabalhar imediatamente, e de uma forma colaborativa através deste tipo de serviço. Alguns exemplos de soluções PaaS são o Google Platform (que inclui uma miríade de opções que vão do App Engine ao Big Data) e o Microsoft Azure.

Software-como-serviço

No modelo SaaS, o fornecedor disponibiliza aplicações prontas-a-usar, ou que podem ser configuradas pelo "consumidor" (a organização-cliente) para disponibilizar aos utilizadores finais. Alguns exemplos de soluções SaaS são o Gmail, DropBox e Microsoft SharePoint.

2.4 Estudo de Caso: OpenStack - uma nuvem IaaS

Esta secção apresenta os conceitos utilizados pelo OpenStack e arquiteturas dos vários serviços existentes no mesmo. A secção 2.4.1 é uma breve introdução ao OpenStack. Na secção 2.4.2 é apresentada a arquitetura do OpenStack, segue-se um desenho mais detalhado das relações entre os módulos, dos seus componentes internos mais relevantes, e das APIs exportadas. Finalmente, a secção 2.4.3 apresenta cada um dos principais componentes do OpenStack (e que correspondem, do ponto de vista do desenvolvimento, a projetos distintos, sendo por isso por vezes assim designados).

2.4.1 Introdução e Conceitos

O OpenStack é uma plataforma de código aberto para construir uma nuvem IaaS altamente escalável. Originalmente desenvolvido pela Rackspace e NASA, o OpenStack organiza numa única infraestrutura componentes de computação, armazenamento e rede, geridos como um todo através de um painel que disponibiliza vistas distintas para os administradores de infraestrutura e os consumidores dos recursos [Opeb].

O OpenStack foi desenhado para correr em *hardware off-the-shelf*, e está principalmente escrito em C, C++ e Python. O OpenStack tem uma arquitetura modular, na qual cada componente (módulo) é responsável por um serviço e exporta uma API: os módulos fundamentais (e que do ponto de vista de desenvolvimento correspondem a diferentes projetos) são: painel de controlo (Horizon), identidade (Keystone), computação (Nova), imagens (Glance), rede (Neutron), armazenamento de objetos (Swift), armazenamento de blocos (Cinder), monitorização (Ceilometer), orquestração (Heat) e base de dados (Trove).

2.4.2 Arquitetura e Conceitos

A Figura 2.7 mostra as relações entre componentes e serviços por eles prestados. Como se pode observar, três destes componentes interagem com todos os outros: o Horizon, que fornece uma visão dos vários serviços disponíveis tanto para o utilizador, como para o administrador; o Neutron que fornece ligações a rede virtuais; e o Keystone, que autentica todos os serviços e utilizadores.

De uma forma muito sumária, e considerando apenas a perspetiva do utilizador, a execução de máquinas virtuais é realizada da seguinte forma: i) o utilizador escolhe o **flavor** (determinando as "capacidades" do seu servidor virtual) e a **imagem** (determinando qual o stack de software – SO, etc. – que quer usar); ii) de seguida, lança a instância em execução. Durante a execução pode, se o desejar, efectuar **snapshots**, e pode **desligar** e arrancar a VM. Quando a VM já não for mais necessária, pode **terminá-la**, deixando de ter acesso à informação que esta continha.

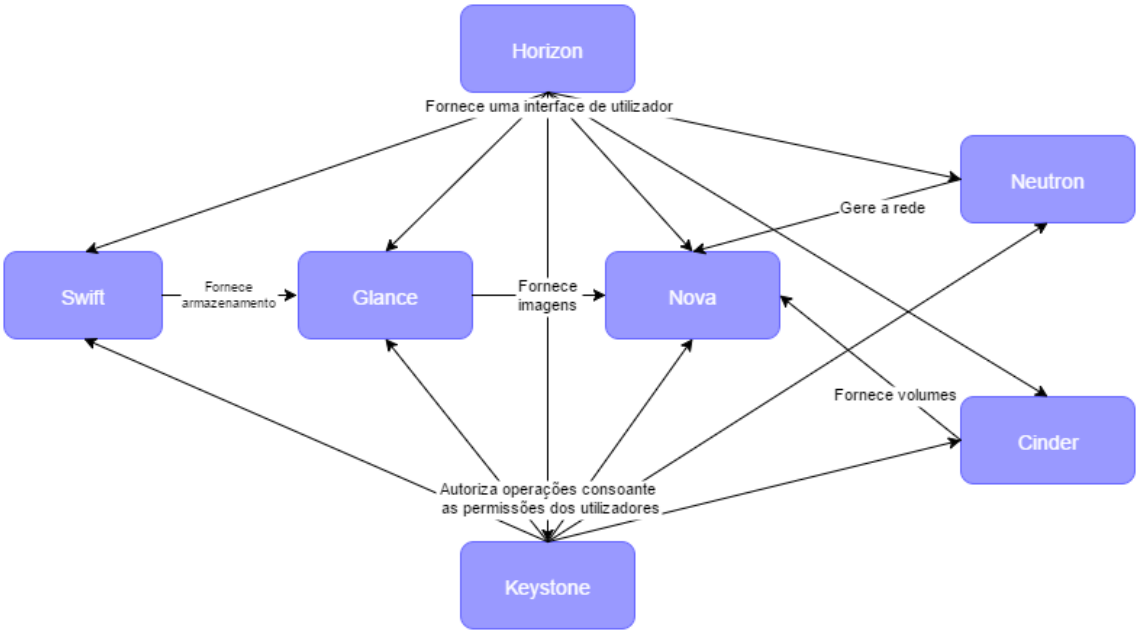


Figura 2.7: Arquitetura Conceptual do OpenStack.

O OpenStack engloba vários componentes que foram concebidos para ser tão independentes quanto possível, de modo que os utilizadores podem instalar apenas um pequeno subconjunto (por exemplo, Compute, Imagem e Networking) e ainda obter uma nuvem utilizável e estável. A comunicação entre os utilizadores finais e serviços podem ir através da aplicação web Horizon, ou através da API do serviço. A Figura 2.8, retirada de [Opea], é uma visão mais detalhada da arquitetura OpenStack e as relações de seu módulo.

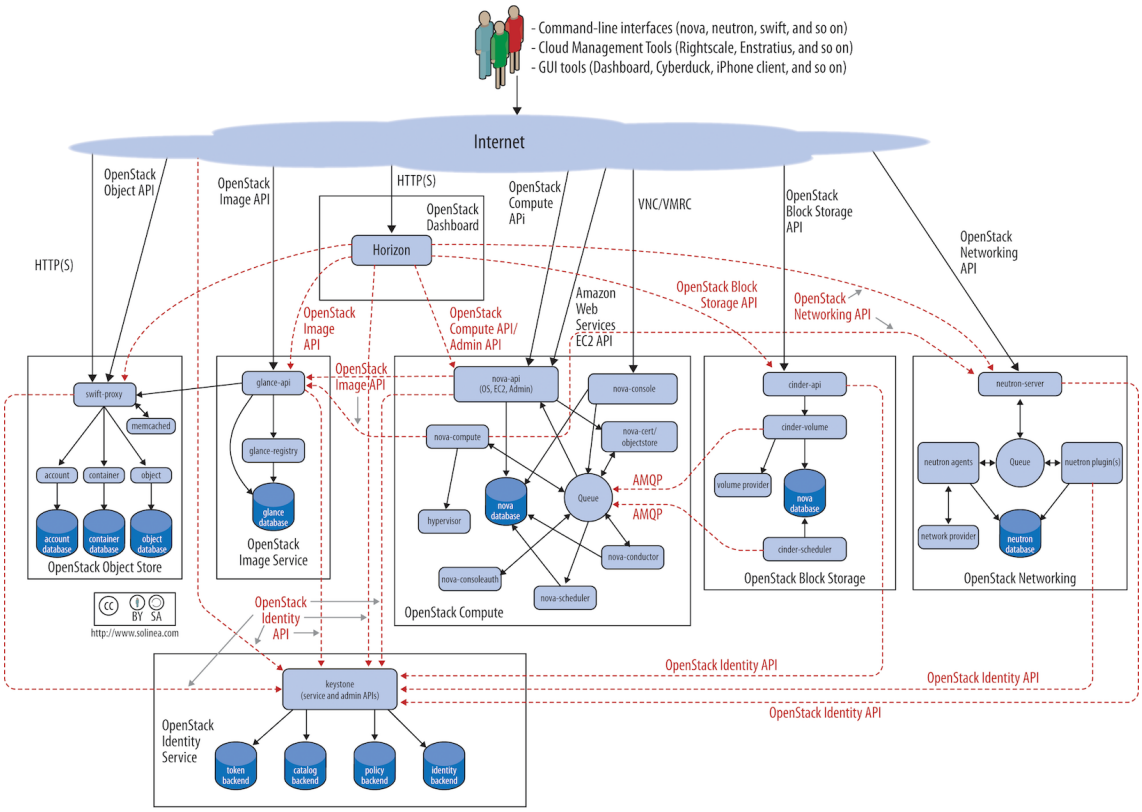


Figura 2.8: Arquitetura mais detalhada do OpenStack.

Uma descrição mais técnica, que para melhor compreensão sugerimos que seja acompanhada seguindo a Figura 2.8, da execução de máquinas virtuais pode ser a seguinte, considerando desde já que o serviço de identidade, **Keystone**, realiza todas as autenticações necessárias a cada etapa: i) o serviço de computação, Nova, que executa instâncias de VMs, começa por reunir os recursos necessários definidos pelo flavor; em seguida, solicita ao serviço de imagens, **Glance** a imagem, criando uma cópia "local" (i.e., num repositório "do Nova") dessa imagem, cópia essa que passa a constituir a **instância** (ou, mais precisamente, o disco de arranque da instância); finalmente, lança a VM em execução. O utilizador tem acesso à consola da VM através do **Horizon**, e a VM pode comunicar com outras e/ou com "a Internet" se a configuração de rede definida no flavor, e criada pelo **Neutron**, o permitir.

Enquanto a instância existir, o estado das sucessivas execuções (shutdowns/boots), snapshots, etc., é mantido pelo Nova no "seu" repositório, sendo removido quando a instância for terminada (removida talvez fosse um termo mais apropriado); as instâncias são por isso designadas **transitórias**. Se o utilizador quiser continuar a dispor, de forma **persistente**, de dados de execução após remoção das instâncias pode consegui-lo de duas formas distintas: configurando no flavor discos adicionais, que serão geridos pelo **Cinder** ou configurando na instância (depois de esta já estar em execução) acesso(s) a repositórios de objetos, mediados pelo **Swift**.

2.4.3 Componentes do OpenStack

2.4.3.1 Painel de Controlo (Horizon)

O Horizon, painel de controlo do OpenStack, é uma aplicação web que fornece uma *interface* gráfica a serviços do OpenStack (tais como o Nova, o Swift, o Keystone, etc.) para realizar tanto tarefas de administração (do próprio OpenStack) como de utilizador, quer sejam mais especializadas (instanciação de máquinas virtuais, atribuição de endereços IP, configuração de controles de acesso) ou de "mero consumo" de serviços, como sejam o arranque e paragem das VMs, acesso à consola, etc..

2.4.3.2 Identidade (Keystone)

O Keystone é um projeto OpenStack, que fornece uma API que possibilita a serviços e utilizadores enviarem pedidos de forma a que o Keystone os possa identificar, O Keystone é fundamentalmente um serviço de autenticação, autorização e acesso. Inclui ainda o catálogo de utilizadores e de serviços (fornecendo os seus *endpoints*), e todas as funcionalidades que implementa são acessíveis através de uma API.

2.4.3.3 Computação (Nova)

O Nova é um serviço absolutamente fundamental, pois oferece serviços de computação, i.e., é o componente encarregue da gestão das instâncias de VMs nos seus múltiplos estados (em criação, arranque, em execução, suspensas, paradas, terminadas, etc.). O Nova suporta diversos hipervisores.

2.4.3.4 Serviço de Imagens (Glance)

O Serviço de imagem (Glance), oferece serviços de pesquisa, registo e entrega (*delivery*) de imagens de disco virtual, cujos formatos podem ser ami (EC2), qcow2 (Qemu/KVM), raw, vhd (Hyper-V), vdi (Virtual Box), vmdk (VMware e outros). As imagens são armazenadas num *object storage*, podem ser acedidos através de uma interface REST, estando registadas numa base de dados.

2.4.3.5 Armazenamento de Blocos (Cinder)

O Cinder é um serviço gestão de volumes que suporta as necessidades de "*block devices*" de outros serviços, como por exemplo, o Nova; como outros serviços do OpenStack, não só tem a sua própria implementação de um sistema de armazenamento de blocos (baseada em discos locais ao nó Cinder, geridos via LVM) como, em alternativa, suporta uma grande variedade de provedores externos: Ceph, GlusterFS, iSCSI, NFS, Sheepdog, etc..

2.4.3.6 Armazenamento de Objetos (Swift)

O Swift é um sistema de armazenamento de objetos (*object store*) que pode ser usado, simultaneamente, de duas formas distintas: para armazenar e gerir imagens de discos virtuais (i.e., *templates*, necessários ao Nova para criar instâncias); e para constituir "objetos de armazenamento" que podem ser acedidos pelas instâncias para nelas armazenarem informação de forma persistente. Tal como o Cinder, o Swift não só tem a sua própria implementação de um sistema de armazenamento de objetos como, em alternativa, suporta uma grande variedade de provedores externos: Ceph, GlusterFS, iSCSI, NFS, Sheepdog, etc.. O sistema nativo de armazenamento de objetos do Swift será estudado em detalhe no próximo capítulo.

2.4.3.7 Rede (Neutron)

O Neutron pode ser descrito como um fornecedor de "rede como um serviço" (*Network-as-a-Service*), pois permite o isolamento entre redes de, por exemplo, inquilinos (*tenants*) distintos da *cloud* OpenStack, ou a criação de segmentos separados para redes de um mesmo inquilino.

2.5 Conclusão

Em um ambiente de HPC, a virtualização é a abordagem mais confiável para fornecer os muitos recursos de computação e armazenamento necessários, dividindo logicamente os recursos fornecidos. A abordagem mais simples para a computação em nuvem é a instanciação de máquinas virtuais, e OpenStack fornece o serviço necessário. O armazenamento está diretamente associada com ambientes HPC, necessárias principalmente para resultados da experiência, por isso, OpenStack também fornece serviços capazes de atender clientes com o espaço necessário.

Apesar de a abordagem OpenStack fazer uso de armazenamento de objetos, Swift, e fá-lo bem, isso aumenta a complexidade da gestão quando o armazenamento de blocos também é necessário. Com isso em mente, uma necessidade inevitável para uma solução de armazenamento de *backend*, que unifica armazenamento de objetos e de blocos, mantendo ou fornecendo novos recursos e desempenho, pode ser considerado como um grande trunfo para o administrador do sistema.

3

Armazenamento no OpenStack

O armazenamento é a segunda infraestrutura mais complexa (a primeira é a rede) do OpenStack. Felizmente, a maioria da complexidade resulta da ausência de documentação adequada - do ponto de vista conceptual: quase todos os documentos, incluindo livros disponíveis comercialmente, são "guias de referência" (de administrador, de utilizador, etc.) ou "guias de instalação". Esperamos, por isso, que esta secção contribua para uma compreensão desta infraestrutura e suas muito variadas opções.

A Figura 3.1 abaixo mostra apenas os módulos relacionados com o armazenamento: o Swift, para objetos e o Cinder para blocos, estes são dois (módulos) fornecedores de armazenamento, enquanto que os módulos Glance, de Imagens e Nova, de Computação são consumidores de armazenamento.

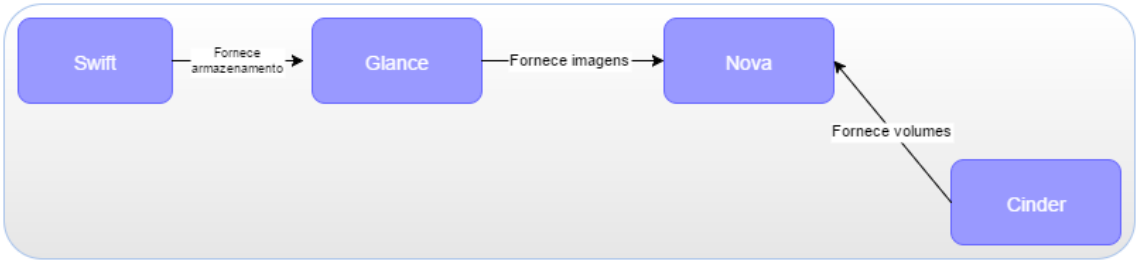


Figura 3.1: Módulos relacionados com o armazenamento.

Os diferentes tipos de armazenamento existem para suportar as necessidades das VMs e, na terminologia do OpenStack, são três: armazenamento de imagens (imagens

de discos virtuais, i.e., dados estáticos incluindo o sistema operativo) a partir das quais as instâncias são criadas; armazenamento persistente (a própria instância, com os discos virtuais que a acompanham) e armazenamento efémero (os dados em memória da instância).

3.1 Armazenamento de Blocos

Certamente como resultado da evolução do OpenStack, que começou simples e tem tido, do ponto de vista da sua complexidade um crescimento exponencial, existem atualmente duas formas distintas de armazenamento de blocos: a mais simples, embebida no módulo Nova e designada *nova-volume*, pode ser descrita como um sistema de armazenamento local, funcionando isoladamente para cada nó/instância computacional Nova; a outra é o já mencionado módulo Cinder, que fornece armazenamento ao nível de uma cloud OpenStack.

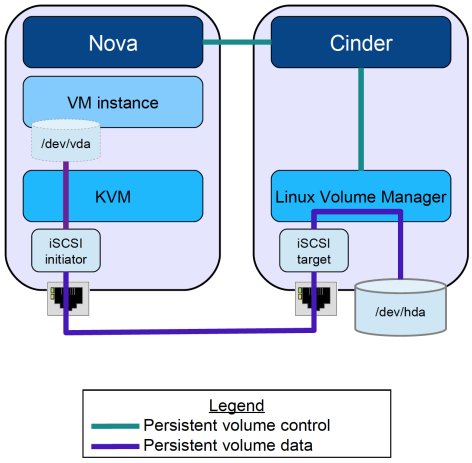
3.1.1 nova-volume

A descrição do *nova-volume*, designado um serviço legado (*legacy*) será breve, por duas razões: em primeiro lugar, a informação de que fomos capazes de reunir (tendo em conta os nossos objetivos e tempo disponível) é incompleta; e, por último, a nossa meta é uma implementação baseada em Cinder.

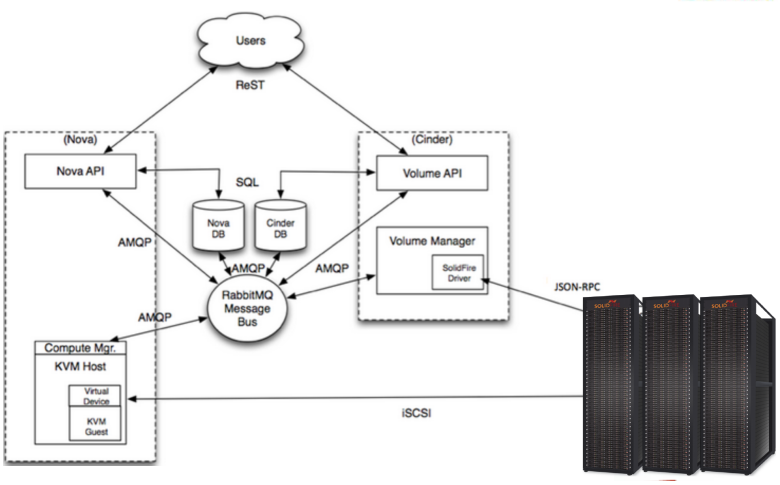
Tanto quanto foi possível perceber, uma configuração *nova-volume* funciona isoladamente para um único nó de computação Nova, usando algum espaço de armazenamento local em discos internos do próprio nó, ou usando volumes (de discos externos) acessíveis usando iSCSI ou AoE (*ATA over Ethernet*). Aparentemente para poder usar armazenamento interno este tem de estar "particionado" usando o LVM; a falta de informação e a inconsistência entre a documentação de diferentes versões reforça a percepção de que a utilização do *nova-volume* como provedor de armazenamento foi abandonada.

3.1.2 Cinder

O Cinder corresponde a uma reformulação substancial da nova-volume (ver Figura 3.2): i) há uma maior separação de papéis, existindo agora módulos como o *cinder-api*, *cinder-volume*, *cinder-schedule*, e *cinder-backup*; ii) cada módulo pode (deve) ser instalado num nó separado; e iii) no cinder-volume, há um desacoplamento adicional entre os aspetos de gestão de volumes e os drivers de armazenamento, podendo-se assim oferecer uma *interface* de gestão que se mantém igual independentemente dos provedores de armazenamento. Há assim "plugins" de backend, como se mostra na Figura 3.2(a), retirado de [Cin], para acesso a infraestruturas SAN baseadas em Fiber Channel (FC), AoE e iSCSI, e para outros provedores de armazenamento, como é o caso do Ceph, GlusterFS, NFS e Sheepdog.



(a) Solução Cinder iSCSI.



(b) Exemplo Cinder utilizando a solução de armazenamento SolidFire.

Figura 3.2: Arquitetura do Cinder.

3.2 Armazenamento de Objectos

3.2.1 Swift

O Swift é um sistema de armazenamento de objetos escalável (até PB de dados), redundante, e com uma arquitetura distribuída composta por *clusters* de servidores comuns (*off-the-shelf*); a sua tolerância a faltas é conseguida à custa da replicação de (partes de) objetos, que são armazenados em vários nós, mantendo a consistência. O aumento da capacidade de um *cluster* Swift faz-se horizontalmente, pela simples adição de novos nós. Em caso de falha de um nó, os dados replicados num outro são utilizados, proporcionando, assim a continuidade de serviço. Todas estas propriedades são de pura responsabilidade do Swift, não tendo em conta quaisquer mecanismos de escalabilidade e tolerância a faltas que possam estar instalados numa camada inferior.

O acesso aos objetos realiza-se em HTTP, utilizando uma API RESTful que inclui as seguintes operações: criar, listar e eliminar recipientes; carregar, descarregar, e listar objetos, e definir as suas permissões com ACLs próprias do Swift. Todos os pedidos incluem o método (por exemplo, GET, PUT, DELETE), a informação de autenticação, a URL de armazenamento de dados e metadados (opcional). Cada caminho de armazenamento contém uma conta de utilizador, o recipiente do objeto e um nome de objeto.

O Swift, é composto por vários componentes, dos quais destacamos:

Servidor proxy Recebe pedidos por duas APIs: a de objetos e a HTTP, e comunica com o módulo Swift. Os pedidos podem ser de uma natureza muito diversificada: envio (*upload*) de ficheiros, operações de atualização de metadados, criação de inquilinos (*tenants*), listagens várias (de ficheiros, inquilinos, etc.). Pode usar a memcache para acelerar as operações.

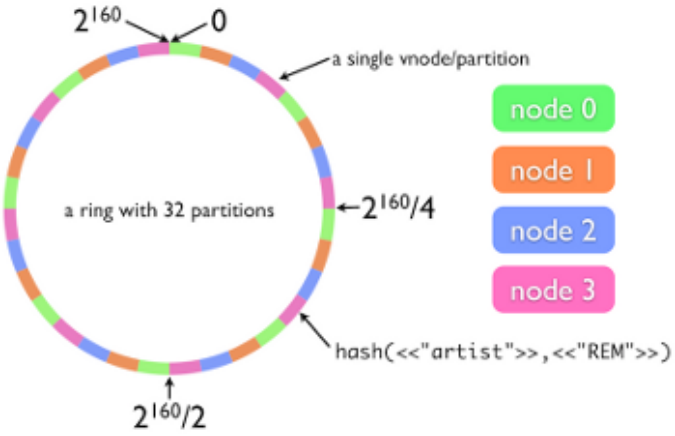
Servidor de contas Responsável pela gestão de contas definidas localmente para o serviço de armazenamento de objetos, pode ser substituído por um outro serviço que use a Web Server Gateway Interface (WSGI), como o serviço de identidade Keystone do OpenStack.

Servidor de Contentores Responsável por implementar a noção de "pastas" (contentores) sobre uma base de dados MySQL, de forma permitir agrupar objetos.

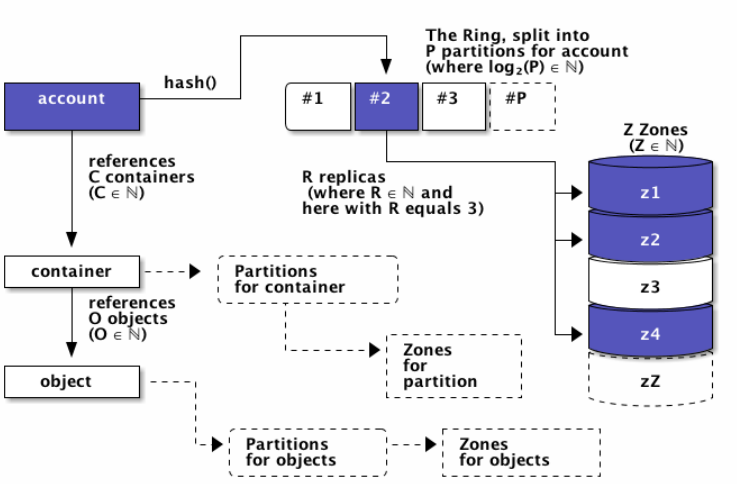
Servidor de objetos Responsável por gerir os objetos, sendo capaz de armazenar e recuperar um objeto dado o seu caminho completo. Um objeto é implementado como um ficheiro local num dos nós de armazenamento. Há várias soluções de backends de armazenamento que estão disponíveis para o Swift, e que vão dos sistemas de ficheiros locais, como o ext4 ou o XFS, a sistemas de ficheiros distribuídos de grande escala, tais como GlusterFS e Ceph, passando por sistemas de ficheiros cliente/servidor, como o NFS.

Serviço de tarefas periódicas Responsável por manter o sistema "saudável", inclui a gestão de recursos (por exemplo, memória virtual) e manutenção da consistência.

Distribuição de Dados O Swift utiliza três estruturas de dados conhecidos como



(a) Anel de *hash* consistente e particionado.



(b) Armazenamento de contas, contentores e objetos.

Figura 3.3: Arquitetura do Swift.

anéis (uma para contas, outra para *containers* e uma terceira para objetos), Figura 3.3(a), retirado de [Swi], para mapear a localização dos objetos e manter a sua coerência. Um anel é um mapa de *hash* entre o *item* de armazenamento e o nó (servidor) que o detém. O anel é calculado com base nos princípios de *hashing* consistente, em que o espaço de resultados de *hash* é dividido em partes iguais, conhecidas como partições, sendo que cada partição pertence a um nó. A partição-destino é calculada aplicando a função de *hash* ao caminho-de-armazenamento e, depois, calculando o módulo entre o resultado anterior e o número de partições existentes.

Replicação de Dados Zonas são espaços de armazenamento isolados entre si, ou seja, uma zona, digamos Z1, não deve depender de outras zonas, Z2 e Z4; uma zona pode ser um disco, servidor ou bastidor. Estando isoladas, uma falta numa zona não se reflete nas outras zonas; no exemplo exibido na Figura 3.3(b), Z2 e Z4 são candidatas para manter réplicas de Z1. O componente responsável por replicar dados é o `swift-proxy`, que quando solicitado para armazenar um objeto, replica-o para todas as zonas adequadas.

Consistência de Dados Caso uma das zonas falhe, o `swift-proxy` não pode replicar novos objetos para essa zona; nesse caso, um conjunto diferente de processos por nó, `swift-account-replicator`, `swift-container-replicator` e ainda `swift-object-replicator`, são chamados para lidar com essa falta e, mais tarde, propagar os dados com rsync. É possível a existência de dados não consistentes, pois a natureza deste *storage* tem como prioridade a disponibilidade dos objetos e não a sua consistência.

Mencionamos finalmente um leque de três serviços: o de "Auditores", que são executados em todos os nós de armazenamento e verificam a integridade dos objetos, contentores e contas; e, caso haja corrupção de dados, colocam-no em quarentena e é inciam o processo de obtenção de uma nova réplica; e o de "Atualizadores", que garantem consistência de contas/contentores e objetos; e o dos processos responsáveis pela eliminação de contas, que procuram periodicamente as contas marcadas para remoção e, quando as encontram, utilizam o serviço apropriado para as remover.

3.3 Fornecedores de Armazenamento *Backend* para o OpenStack

3.3.1 Network File System

O NFS, Network File System [San86], é provavelmente o sistema de ficheiros distribuídos mais conhecido, e segue uma arquitetura cliente-servidor: o servidor NFS exporta a partir da sua sub-árvore da hierarquia do SF "local" para sistemas "remotos" dos clientes que, em seguida, montam a sub-árvore exportada num ramo do seu sistema de ficheiros local. Todas as operações E/S solicitadas pelos clientes são executadas no servidor, e todas as mudanças que acontecem na zona exportada pelo NFS são observadas (ainda que possam ser ao fim de algum tempo) pelos clientes que a partilham como se ocorressem no seu próprio SF local.

A versão do NFS mais amplamente utilizada é a 3, baseada num único servidor, o

que penaliza tanto a disponibilidade como o desempenho; no entanto, a versão 4.1 (a.k.a. pNFS) oferece uma espécie de *striping* sobre vários servidores. A utilização da *cache* pode ser usado, nos clientes para aumentar o desempenho e minimizar o tráfego cliente/servidor, mas pode conduzir a incoerências temporárias entre dados dos clientes e, por isso, deve ser usado com cuidado.

O NFS pode ser usado no OpenStack como *backend* tanto do Cinder, usado portanto para oferecer armazenamento de blocos, como pelo Swift, sendo aí utilizado para suportar os "objetos" que depois serão acedidos pelos outros módulos, como por exemplo, o Glance e o Nova. Os aspetos distribuído e partilhado são de especial importância para permitir a migração sem cópia (das imagens) de instâncias/VMs de um nó (Nova) para outro.

3.3.2 GlusterFS

O GlusterFS ([Oli07][GLM13][Glub][Glua]) é um sistema de ficheiros distribuídos de código aberto. O GlusterFS, oferece uma solução que escala horizontalmente, ideal para armazenar grandes quantidades de dados; também fornece várias APIs para acesso ao seu armazenamento, incluindo GlusterFS, NFS, CIFS (SMB) e HTTP.

Conceitos e Arquitetura

O GlusterFS armazena dados em vários nós de armazenamento (servidores) em que cada nó contém uma ou mais entidades nomeadas *bricks*; um *brick* é (geralmente) um disco ou partição de disco que está formatado sob algum sistema de ficheiros local (ext3, XFS, etc.) e, em seguida, é "exportado" como um par <*host*:/diretório>.

Os dados que entram ou saem de um *brick* são processados por um conjunto de "tradutores", bibliotecas dinâmicas que podem ser carregadas por clientes ou servidores para fornecer alguma funcionalidade adicionada. Um sub-volume é um *brick* depois de ser processado por um tradutor, e um volume é uma agregação de sub-volumes depois de serem processados por todos os tradutores. O cliente é a máquina que monta o volume (clientes também podem agir como servidores).

Ao contrário de outros sistemas de ficheiros distribuídos, o GlusterFS não tem servidores exclusivos para gestão de metadados, localiza ficheiros através de um algoritmo: dando o nome do ficheiro e o caminho, um servidor de armazenamento utiliza uma função de *hashing* para localizar o ficheiro. Esta estratégia, assegura que o desempenho do sistema escala linearmente, e elimina a necessidade de sincronização de metadados.

O cliente GlusterFS, é incomum, no que respeita a sistemas de ficheiros distribuídos, pois é responsável por muitas coisas: gestão de volumes, escalonamento do E/S, localização de ficheiros, armazenamento de dados em *cache*, etc. Os clientes executam um processo (Glusterfsd) para ajudar *mounts* de volumes GlusterFS a nível de utilizador (FUSE), nas suas árvores de SF local.

Existem vários tipos de tradutores: `Cluster`, que especifica como distribuir dados,

réplicas e operações E/S; `Debug`, que fornece uma interface e os dados estatísticos de erros e depuração dos mesmos; `Encryption`, que oferece um protocolo de criptografia de armazenamento simples; `System`, que permite o acesso ao sistema; `Performance`, que permite que o sistema se adapte a diferentes cargas de trabalho e operações de E/S (fornece operações FS assíncronas, *cache* de E/S e *write-behind*); `Scheduler`, que fornece escalonadores de E/S que determinam a forma de distribuir as operações de escrita; `Protocol`, que fornece autenticação de cliente/servidor e respectivas comunicações; `Storage`, que especifica os padrões de armazenamento de dados e de acesso para o sistema de ficheiros local; `Bindings`, fornecendo extensibilidade; `Features`, que fornece funcionalidades adicionais, como mecanismos de *locks*, *filtros*, etc.

O tradutor `Cluster` é particularmente interessante, pois gere a distribuição de dados e/ou replicação sob cinco configurações distintas: Replicação de Ficheiro Automática (Automatic File Replication ou AFR), Tabela de Dispersão Distribuída (Distributed Hash Table ou DHT), Acesso Não Uniforme a Ficheiros (Non Uniform File Access ou Nufa), Stripe, e por último Alta Disponibilidade (High Availability ou HA) .

O tradutor `Cluster` com AFR, comporta-se como RAID-1: replica dados em vários sub-volumes, mantendo-os sincronizados e, no caso de uma falha no servidor, serve pedidos fornecendo réplicas. Pedidos de leitura são enviados para todos os nós de armazenamento para um melhor desempenho. A AFR também fornece recuperação automática, quando o servidor que sofreu uma falha, fica novamente contactável, todos os ficheiros e diretórios são atualizados para suas últimas versões.

O tradutor `Cluster` com uma DHT utiliza um algoritmo de *hashing* consistente: distribui ficheiros submetendo os seus nomes a uma função de *hashing*. Tem três opções: `lookup-unhashed`, `min-free-disk` e `sub-volumes`. O `unhashed-lookup` irá forçar o tradutor a enviar chamadas de pesquisa para todos os sub-volumes. O `min-free-disk` apenas cria ficheiros no volume alvo (definido pela função de *hashing*) até o seu espaço de armazenamento disponível ser menor do que algum valor estabelecido (por omissão: 10 %). O tradutor `sub-volumes` retorna um erro se houver apenas um sub-volume disponível. O tradutor com DHT, é geralmente configurado juntamente com AFR.

O tradutor `Cluster` como NUFA favorece armazenamento local sobre armazenamento remoto; isso pode resultar em melhor desempenho em ambientes como HPC. Tem as opções `lookup-unhashed` e `sub-volumes` assim como DHT, acrescentando `local-volume-name` para designar o volume local.

`Stripe` comporta-se como RAID-0: um arquivo é distribuído em vários nós, com um tamanho de bloco de 128 KB. Todos os nós constituem um espaço de nomes completo, de modo que a leitura e a escrita pode ser executado em paralelo, o que resulta num desempenho melhorado.

Finalmente, o tradutor HA, fornece *failover* entre dois volumes.

GlusterFS e o OpenStack

O GlusterFS pode ser utilizado como um *backend* de armazenamento para o OpenStack de várias maneiras: ele pode ser integrado com o Glance para armazenar imagens, com o Cinder a fornecer volumes para VMs, e com o Swift. Além disso, ele pode ser integrado com o Keystone, para autenticação e autorização.

3.3.3 Sheepdog

O Sheepdog é um sistema de armazenamento distribuído desenvolvido especificamente para fornecer volumes ou contentores de objetos a VMs a executar em ambientes KVM, sendo a sua escalabilidade viável até centenas de nós de armazenamento.

Arquitetura

A arquitetura simétrica do Sheepdog não necessita de um servidor central, pois todos os nós são responsáveis por armazenar objetos, e gerir o *cluster*. Os componentes que compõem a arquitetura (Figura 3.4, retirado de [She]) do Sheepdog são:

- Gateway: Este componente recebe os pedidos E/S vindos do cliente, juntamente com os respetivos parâmetros necessários para as operações que pode realizar (identificador global do objeto, *offset*, tamanho e tipo de operação) pelo *driver* QEMU, calcula os nós de armazenamento que irão armazenar o objeto, calculo feito através de uma função *hashing* consistente.
- O Gestor de Objetos (ou Object Manager): Este componente recebe pedidos E/S do Gateway e executa as operações no disco local.
- Gestor do Cluster (ou Cluster Manager): Componente responsável por gerir os membros do *cluster*, o Gestor do Cluster possui a responsabilidade de adicionar ou remover membros quando há alteração do *cluster* face a comandos do administrador ou face a falhas, sendo ainda responsáveis por gerir operações que necessitem de um consenso geral dos nós de armazenamento.
- Driver QEMU para armazenamento baseado em blocos: O lado cliente do sistema, divide a imagem da VM em objetos (que por omissão possuem uma dimensão de 4 MB) e armazena-os através do Gateway.

Objetos

Os objetos são de tamanho variável e possuem um identificador global; fornecendo este identificador, é possível realizar operações de leitura e escrita. A camada de armazenamento de objetos implementa vários tipos de objetos:

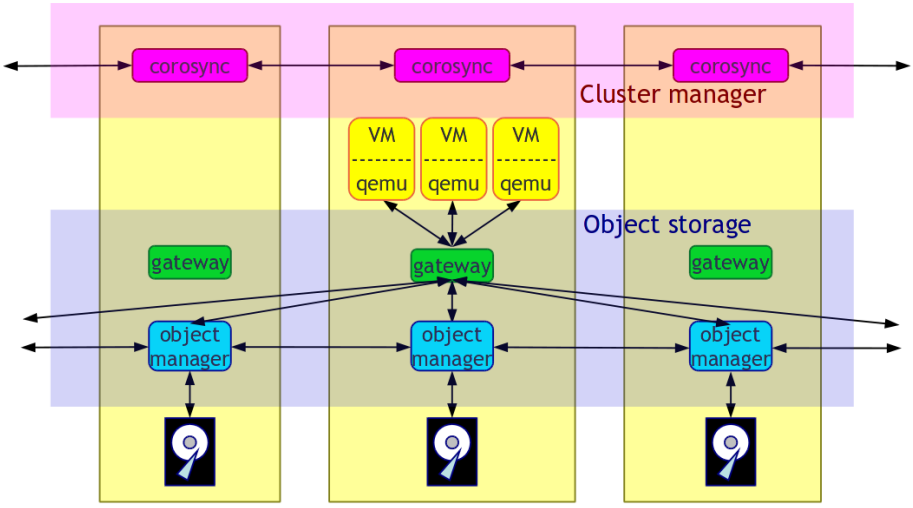


Figura 3.4: Arquitetura Sheepdog.

- Objeto de Dados (ou Data Object): Objetos que armazenam o conteúdo dos discos virtuais das VMs.
- Objeto de Imagem de Disco Virtual (Virtual Disk Image Objet ou VDI Object): Objetos que armazenam metadados dos discos virtuais das VMs (nome de imagem, tamanho, identificadores dos objetos de dados, etc).
- Objeto de estado da VM (VM State Object): Objetos que armazenam o estado atual da VM, sendo usado para carregar o estado da VM na inicialização, resumo, na criação de snapshots, e sendo armazenado o estado no encerramento e pausa da VM.
- Objeto de atributo da imagem de disco virtual (VDI Attribute Object): Funciona como extensões de ficheiros, tendo como estrutura pares chave-valor.

O algoritmo utilizado pelo Sheepdog para distribuir os objetos e réplicas pelo armazenamento é o *hashing* consistente. Em termos de replicação, o Sheepdog assume que em qualquer instante só existe um escritor, pelo que não existe a possibilidade ocorrer colisões de escrita; por isso, clientes podem enviar operações de escrita em paralelo, que não existe a possibilidade de ambos os escritores estarem a escrever nos mesmos blocos ou objetos, deixando-os incoerentes.

A execução do fluxo de operações de escrita no sistema de armazenamento Sheepdog é como se segue: 1º) o Gateway recebe um pedido de escrita, calculando através de uma função de *hashing* consistente, o resultado será um vetor de nós para o qual o Gateway irá reencaminhar o pedido; 2º) a operação é enviada em simultâneo para todos os nós (tendo em conta que existe replicação), e quando a escrita é concluída em todos os nós, a operação também é dada como concluída. Em leituras, a localização dos objetos é calculada pela função de *hashing*, sendo a operação reencaminhada para um dos nós do vetor resultante.

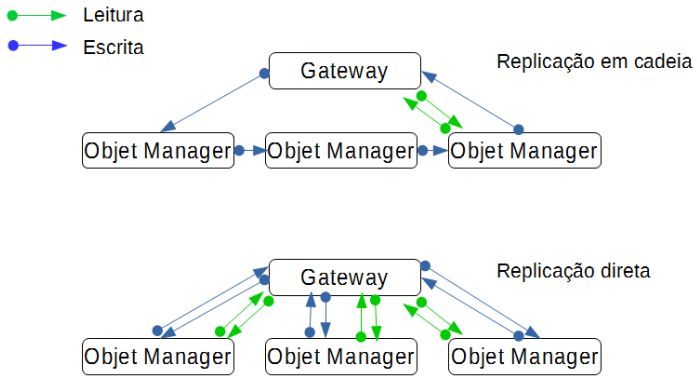


Figura 3.5: Replicação em cadeia e replicação direta.

Replicação

Um pedido de escrita de um objeto chega ao Gateway, após o qual, este é responsável por definir o(s) destino(s) onde o objeto vai ser armazenado. O Gateway fica responsável por assegurar que todas as réplicas do objeto são escritas em disco antes de dar a operação como concluída.

No Sheepdog a replicação é feita de duas formas: pode-se utilizar uma estratégia de replicação em cadeia, em que o Gateway envia o pedido para o primeiro Gestor de Objetos, este envia para o segundo e assim sucessivamente até ser enviado para todos responsáveis por armazenar o objeto; a segunda estratégia implica o Gateway enviar diretamente o pedido para todos os Gestores de Objetos, tomando partido do paralelismo inerente (isto só acontece para escritas, leitura é sempre feita para um).

É ainda possível introduzir (tal como no Ceph) mecanismos de resiliência em forma de objetos de paridade, permitindo utilizar mais eficientemente o espaço de armazenamento disponível.

Recuperação

O histórico de filiação dos nós é guardado num histórico que no Sheepdog é referido como "Época", ou seja, são guardadas várias versões do estado do *cluster*. Quando a Gateway faz um pedido de E/S para um dado nó e as épocas entre ambos são diferentes, os pedidos (já na fila de espera do nó) falham, é necessário que o Gateway repita os pedidos até que as épocas coincidam. Assim, é possível obter uma consistência forte para objetos replicados. A Gateway pode ter o mesmo comportamento caso encontre nós offline ou falhados.

O processo de recuperação é implementado pelo Sheepdog dando como prioridade a recuperação do estado dos nós, face a fornecer o serviço de armazanamento. Se o QEMU enviar um pedido E/S para objetos que ainda não recuperaram, esses pedidos são bloqueados até que os objetos estejam na época atual.

O protocolo de recuperação segue o seguinte procedimento:

- O Gestor de Objetos recebe todos os identificadores dos objetos no *cluster*;
- Calcula qual o sub-conjunto de objetos pelo qual é responsável por armazenar;
- Cria uma lista contendo os identificadores pelo qual é responsável por armazenar;
- Envia um pedido de leitura para obter os objetos da lista, sendo o pedido enviado para o Gestor de Objetos que continha o objeto na época anterior;
- Armazena o objeto como forma de ficheiro na diretoria da época correspondente.

Journal

O Gestor de Objetos é responsável por armazenar os objetos (em forma de ficheiro) para o disco local. Operações de escrita ao discos locais são executadas posteriormente, sendo prioritário a escrita no *journal*; a utilização do *journal* serve para registar as modificações feitas ao armazenamento de uma forma atómica, de modo a prevenir atualizações parciais (o que é possível quando introduzimos falhas). Isto aplica-se aos objetos VDI, sobretudo pois se os objetos VDI (os metadados) são parcialmente atualizados, perdemos um estado consistente, o que pode causar uma falha total do armazemento em questão (disco ou discos virtuais).

Gestor de *Cluster*

O Sheepdog utiliza um componente chamado Gestor do *Cluster*, mais propriamente utiliza uma aplicação chamada Corosync Cluster Engine, este tem como responsabilidade verificar a filiação dos vários nós ao *cluster* e o estado de execução de cada um; inclui a adição de nós ou a sua exclusão, tendo também que garantir um consenso maioritário para operações tais como criação ou atualização de objetos VDI, *snapshots*, etc.

Driver QEMU

Abstracções do lado do cliente:

- A abstração baseada em blocos é desenvolvida para VMs que utilizem hipervisores KVM com QEMU para emulação de hardware, no entanto, também disponibiliza armazenamento como dispositivos SCSI (identificados por sda, sdb, etc); suporta ainda funcionalidades avançadas tais como *snapshot*, clonagem e imagens esparsas. O Sheepdog fornece imagens esparsas, isto significa que um volume criado na VM, irá criar uma imagem respetiva no lado do *cluster*, mas esta não aloca necessariamente todo o seu armazenamento no *cluster*, pelo contrário, apenas objetos escritos alocam o espaço necessário em disco;
- A abstração baseada em contentores de objetos é desenvolvida com API compatível ao OpenStack Swift e Amazon S3, sendo por isso utilizada como uma solução alternativa de armazenamento *backend* para estas tecnologias.

Sheepdog e o OpenStack

Sendo muito semelhante ao Ceph (excluindo a camada de FS), o Sheepdog permite integrar com o OpenStack Swift e Cinder, fornecendo armazenamento não partilhado em forma de contentores ou volumes.

3.3.4 Ceph

O Ceph [Wei07b] é um sistema distribuído para o armazenamento de objetos, e inclui também um sistema de ficheiros (ainda experimental). É em código aberto, e foi desenhado para proporcionar escalabilidade, desempenho e confiabilidade, sendo também integrável com os vários serviços de armazenamento do OpenStack, nomeadamente com os serviços de armazenamento de blocos e objetos, como se pode ver na Figura 3.6.

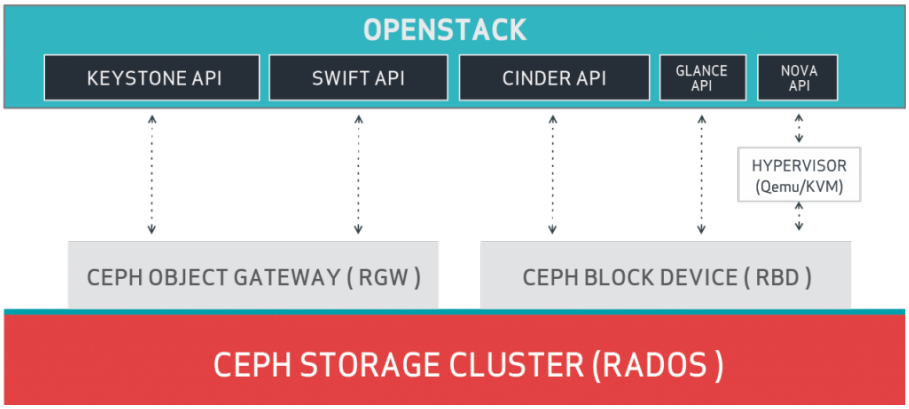


Figura 3.6: Integração do Ceph com o Openstack.

Visão Geral

A arquitetura do sistema de armazenamento Ceph está representada na Figura 3.7 [Cep] e, deixando de lado a parte ainda experimental do sistema de ficheiros CephFS, podemos ver que é fundamentalmente formada por duas áreas: a dos clientes e a dos servidores. A área dos servidores tem dois componentes principais: o núcleo, RADOS (Autonomic Distributed Reliable Object Storage) [Wei07a], que é o serviço responsável pelo armazenamento de objetos, formado por um conjunto de nós que correm os componentes de armazenamento (Object Storage Daemons, OSD), e os monitores; e um serviço de metadados, MDS (Metadata Server) que é usado unicamente pelo CephFS e pode ser implementado como um outro *cluster* de servidores, agora de metadados, que gere um espaço de nomes, mantém a segurança, consistência e coerência.

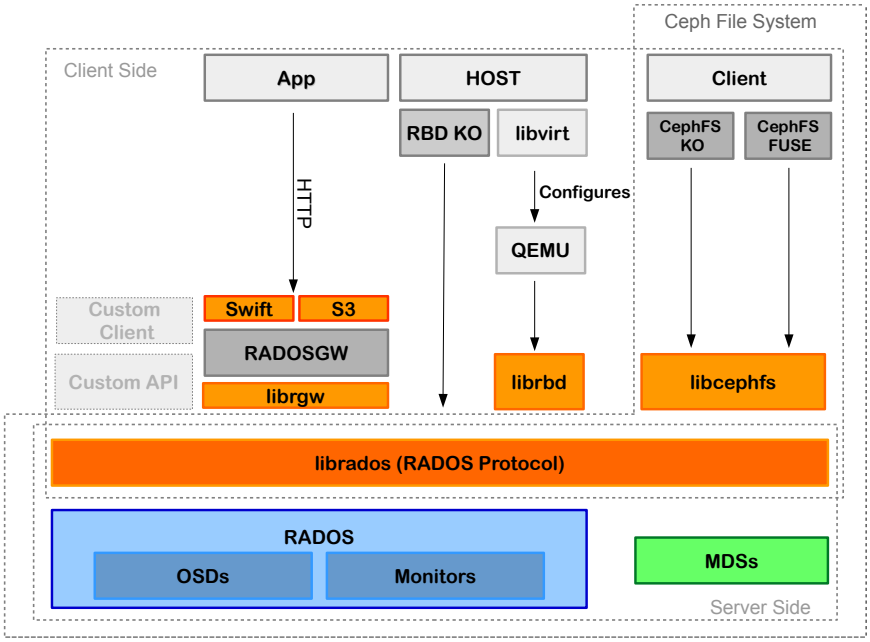


Figura 3.7: Arquitetura Ceph.

Os clientes do serviço "object storage" comunicam com o RADOS usando um protocolo que, do lado do serviço RADOS está implementado pela biblioteca librados. Um caso de uso típico será recorrer ao RADOS para suportar volumes (discos) que o sistema-cliente (tipicamente um servidor x86) formata e usa como se de um disco local se tratasse; isso consegue-se instalando no núcleo do SO do cliente o módulo RBD (RADOS Block Device). Se por acaso, o sistema utilizar um sistema de vertualização, tal como por exemplo Qemu/KVM, podemos usar objetos RADOS para suportar as imagens das VMs; para isso basta configurar o Qemu para utilizar a biblioteca librbd, como mostra a Figura 3.7.

Na Figura 3.8 [Cep] exibe-se uma possível instanciação do Ceph: um conjunto de clientes usa o sistema de ficheiros CephFS, usando a interface Linux Fuse (e não o módulo de kernel). Os pedidos de acesso ao SF são inicialmente dirigidos ao *cluster* MDS para que este resolva a tradução entre nomes simbólicos e objetos; quando o objeto-alvo é selecionado, o cliente acede diretamente ao objeto, que estará segmentado (um valor por omissão é usar segmentos de 4 MB) estando os diferentes segmentos armazenados em nós distintos do *cluster* RADOS, dando-se assim uma oportunidade ao paralelismo para acelerar o acesso aos objetos.

O RADOS Gateway (RGW ou RADOSGW), é uma interface com uma API RESTful que permite o acesso a objetos armazenados no Ceph usando protocolos sobre HTTP; de momento estão disponíveis interfaces de acesso ao Ceph compatíveis com o OpenStack Swift e o Amazon S3.

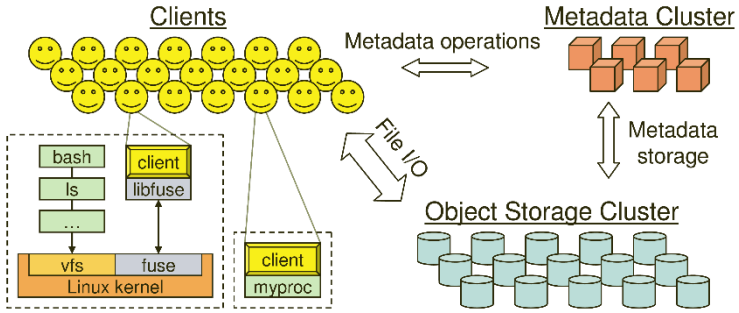


Figura 3.8: Arquitetura do CephFS.

Breve descrição do funcionamento interno do RADOS

O funcionamento detalhado do RADOS será descrito mais tarde, em 3.3.6, já que o Ceph é, podemos desde já adiantar, o sistema escolhido para os nossos testes; mas, para efeitos de comparação das suas funcionalidades e limitações com as dos outros candidatos, deixamos aqui algumas notas adicionais.

Um *cluster* RADOS, é capaz de suportar dinamicamente a sua expansão, por adição de nós, e também a sua contração, pela razão oposta; em qualquer dos casos, o sistema pode reequilibrar-se redistribuindo os objetos pelos nós.

Quando um OSD é adicionado ou removido do *cluster*, essa ação força os monitores a atualizar o mapa do *cluster* (descrito mais detalhadamente na página 43), e por conseguinte, a distribuição de objetos e das suas réplicas muda. Os clientes Ceph não têm conhecimento sobre a localização dos objetos; em vez disso, cada cliente, OSD, MDS ou monitor, executa um algoritmo de localização, designado CRUSH, que mapeia objetos nas suas respectivas localizações (ver em mais detalhe "Pools, grupos de colocação e objetos", em 3.3.6).

A integridade dos dados é assegurada pelo Ceph realizando periodicamente operações de scrubbing sobre a camada de armazenamento de objetos; o Ceph gera catálogos para objetos primários e para as respectivas réplicas e compara-os para verificar se algum objeto está ausente ou se existem diferenças de conteúdo. Existem dois tipos de verificação de integridade : operações "leves", definidas por omissão com uma periodicidade diária, que verificam a dimensão dos objetos e os seus atributos; e operações "profundas", definidas por omissão com uma periodicidade semanal, que verificam a integridade dos objetos e usando checksums.

3.3.5 Uma Análise Preliminar de quatro Fornecedores de Armazenamento Integrável com o OpenStack

Baseamos a avaliação e a taxonomia utilizada, no que foi apresentado na tese de doutoramento[Lop09], e na sequência deste estudo, tomamos o Modelo de Referência para Arquiteturas de Gestão de Dados (Data Management Architectures ou RM-DMA) e apresentamos as camadas que usamos para caracterizar cada sistema de arquivos discutidos

nesta secção (NFS, CephFS e GlusterFS).

Camada de Sistema de Ficheiros

A camada de sistema de ficheiros (CSF) refere-se a apresentar o *software* do sistema que implementa as abstrações de ficheiros e diretórios mapeando-os para blocos.

Instalação : Em relação à arquitetura do sistema de ficheiros, existem apenas dois tipos: local (por exemplo, ext3) e distribuídos (por exemplo, NFS). Em um SF local, ficheiros são acessíveis apenas através da camada do SF local. Um SF distribuídos é implementado de uma forma que a camada SF é distribuído entre vários nós, fornecendo acesso remoto a ficheiros.

Para efeitos desta secção, seguindo os conceitos local e distribuído, vamo-nos concentrar apenas em SF distribuídos.

Função: Este atributo divide SFDs onde alguns podem ter nós com uma arquitetura típica cliente-servidor (por exemplo, AFS, PVFS), e onde os outros podem ter todos os seus nós na qualidade de pares (por exemplo GFS). Assim, dizemos que um SFD assimétrico tem um ou mais nós com funções de servidor (tais como servidores de metadados ou dados) e outros com funções complementares (clientes do SF) com diferentes funções. Um SFD simétrico tem todos os seus nós a executar com as mesmas funções.

Arquitetura do Particionamento de Serviços: Particionamento de serviços diz respeito à instalação de serviços dentro nós; implica que os serviços de servidor FS estão espalhados entre vários nós, ao contrário da abordagem monolítica, onde um nó de servidor único executa todos os serviços do sistema de arquivos. A abordagem particionado O particionamento pode ser ainda dividido numa abordagem: heterogénea e homogénea. A arquitetura de serviços homogénea implica que todos os nós do servidor de executar os mesmos serviços de servidor do SF. A arquitetura de serviço heterogéneo tem vários nós que executam conjuntos distintos de serviços de servidor do SF.

Escalabilidade do Serviço: Escalabilidade é o atributo que caracteriza a capacidade de um sistema de ficheiros para adicionar mais nós de modo a melhorar o desempenho, isto é crucial para um ambiente de nuvem, por isso mesmo, nós também utilizamos este atributo para caracterizar os SF. Um SF pode ser parcialmente escalável se suportar somente a adição de servidores de dados enquanto não suportar a adição de múltiplos servidores de metadados (por exemplo PVFS), totalmente escalável (por exemplo GFS), ou pode não suportar ambos (por exemplo, NFS).

A tabela seguinte 3.1 ilustra a caracterização de um CSF para diferentes SFDs apresentados nesta secção.

Camada de Armazenamento de Objectos

A camada de armazenamento de objetos (CAO) fornece abstrações de alto nível do armazenamento.

	Camada de Sistema de Ficheiros			
	Deployment	**Roles**	**Particionamento**	**Escalabilidade**
NFSv3	Distribuído	Assimétrico	N/A	Nenhum
Ceph	Distribuído	Assimétrico	Heterogéneo	Total
GlusterFS	Distribuído	Assimétrico	Homogêneo	Total
Sheepdog	—	—	—	—

Tabela 3.1: Classificação da camada de sistemas de ficheiros para o NFS, Ceph e GlusterFS.

Instalação: Como na CSF, na CAO a instalação também pode ser local (centralizada) ou distribuída. Na instalação local, centralizada, o controlo e o fluxo de dados são relacionados apenas a um nó. Na instalação distribuída, o controlo e o fluxo de dados são distribuídos por vários nós.

Particionamento: O particionamento é o atributo que caracteriza se os nós de servidores separados implementam tipos distintos de objectos, ou se cada nó implementa o mesmo conjunto de tipos de objetos (sendo estes dados tipos de objectos, metadados, etc).

Escalabilidade: Este atributo caracteriza a capacidade do SF para crescer na camada de armazenamento de objectos. Pode aumentar o número de servidores de armazenamento de objetos e melhorar as capacidades de subsistema (como largura de banda, tolerância a falhas, capacidade).

Uma caracterização dos diferentes SFDs é ilustrado em baixo (Tabela 3.2).

	Camada de Armazenamento de Objectos		
	Deployment	**Particionamento**	**Escalabilidade**
NFSv3	—	—	—
Ceph	Distribuído	Homogéneo	Total
GlusterFS	Distribuído	Homogéneo	Total
Sheepdog	Distribuído	Homogéneo	Total

Tabela 3.2: Classificação do sistema de armazenamento para o Ceph e o GlusterFS.

Camada de Acesso ao Armazenamento

A camada de acesso de armazenamento refere-se aos modelos compartilhados e distribuídos de armazenamento. Os dois paradigmas de partilha de armazenamento são: particionado e partilhado. No armazenamento particionado, todos os nós acedem a conjuntos disjuntos de recursos de armazenamento; enquanto em armazenamento compartilhado, os nós acedem ao mesmo conjunto de recursos de armazenamento.

No SF Ceph, os ficheiros são distribuídos entre objetos, os objetos são agrupados, e depois estes grupos são distribuídos entre OSDs. Os clientes podem aceder ao mesmo ficheiro, ou seja, ao mesmo objeto no mesmo OSD. Assim, Ceph fornece armazenamento compartilhado. Caso seja usado apenas o *cluster* de armazenamento do Ceph (excluímos

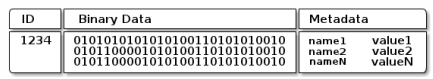
assim o *cluster* de metadados), neste caso, o cliente RADOS Gateway faz a alocação dos objetos, e os clientes RADOS Block devices acedem a conjuntos distintos. O GlusterFS sendo um SFD, oferece armazenamento compartilhado, vários clientes podem assim aceder ao mesmo *brick* para ler o mesmo ficheiro. O Sheepdog é muito estrito, limitando o acesso ao seu armazenamento estritamente a recursos distintos.

3.3.6 Descrição Detalhada do Sistema Candidato

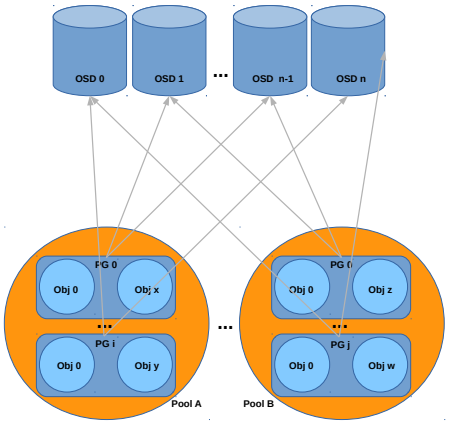
Nesta secção, são descritas em detalhe as várias estruturas de dados, mecanismos e protocolos utilizados pelo Ceph.

Repositórios, Grupos de colocação e Objectos

Vários tipos de dados são correlacionados aos objetos nativos do Rados, são eles: objetos Swift/S3, dispositivos de bloco e ficheiros. O objeto nativo do RADOS, é capaz de armazenar um identificador, dados e metadados com pares de valores-chave (veja a Figura 3.9(a), retirado de [Cep]). Cada objecto, é armazenado no sistema de ficheiro local do OSD, sobre a forma de ficheiro de 4 megabytes (por omissão). Os vários clientes Ceph, são responsáveis pela conversão (particionamento) do objeto cliente para os objetos do RADOS. Cada objeto cliente, é particionado num grande número de objetos Rados, proporcionando assim uma solução semelhante ao RAID. O processo de particionamento pode ser configurado alterando o tamanho da *stripe unit*, o valor da *stripe count* e o tamanho do objecto.



(a) Composição dos objetos nativos do RADOS.



(b) Repositórios, grupos de colocação e objects do RADOS.

Figura 3.9: Composição lógica dos dados do *cluster*.

A fim de diminuir a quantidade de metadados armazenados, Ceph não mapeia cada objecto, em vez disso, objetos são agrupados em estruturas chamadas grupos de colocação (*placement groups* ou PGs) através de uma função de *hashing* que recebe como valor o identificador do objeto e um valor que limita o resultado máximo desta função; por

fim, é utilizado o algoritmo CRUSH, que recebe o identificador do grupo, e que mapeia o grupo objetos a um grupo de OSDs onde irão ser armazenadas todas as cópias do objeto. O conceito de grupo de colocação, baseia-se exclusivamente na redução da quantidade de metadados armazenados no mapa de agrupamento e respetiva gestão. Por exemplo, se temos um dispositivo de bloco de 500 GB teremos 128000 objetos de 4MB para mapear, mas introduzindo o conceito de grupos o mapeamento é reduzido para a quantidade existente destes grupos, que existem em número muito mais reduzido (por exemplo, 128 PGs).

Os dados do *cluster* RADOS são decompostos da seguinte forma: o tamanho da unidade do *cluster* é o objeto; muitos objetos formam um grupo de colocação; e muitos grupos formam um repositório (ou *pool*). Um *cluster* pode ter vários repositórios (por exemplo, separadas para bloco e armazenamento de objetos), cada uma com configurações diferentes (com diferentes níveis de replicação, diferentes regras de distribuição etc).

Em caso de falha, o Ceph fornece dois métodos principais para assegurar a proteção e recuperação de dados dentro dos seus repositórios: replicação e cálculo de objetos de paridade (na terminologia do Ceph, intitula-se *erasure coding*). Replicação como já foi descrito, fornece réplicas para os objetos distribuídos por vários OSDs. A mecânica de um repositório com cálculo de objetos de paridade é bastante diferente do método de replicação. A técnica padrão utilizada no Ceph é uma implementação clássica de teoria codificação para correção de erros, chamada codificação de Reed-Solomon; esta técnica, especifica que cada objeto é armazenado em $K + M$ pedaços distribuídos através de $K + M$ OSDs (sendo K o número de objets de dados, e sendo M o número de objetos de paridade), desta maneira, o conjunto pode tolerar a perda de M OSDs sem perder dados. Dado que objetos de clientes RBD e RGW, podem ter terabytes de dimensão, pode ser impraticável calculá-los em memória, os clientes Ceph dividem estes objetos em objetos do RADOS de menores dimensões (por exemplo, 4MB) e são estes objeto menores que serão alvo dos cálculos por parte dos OSDs. O mecanismo de cálculo de paridade, está localizado no OSD e é realizado para leituras e escritas pelo OSD primário.

Um repositório de objetos, mais conhecido por *pool*, pode ser configurado de forma a proporcionar um melhor desempenho; o administrador do sistema, pode definir um repositório para servir como uma camada de *cache* para uma camada de armazenamento (referido na Figura 3.11, retirado de [Cep]). Ainda, um repositório com *erasure coding*, pode servir de armazenamento com uma camada de *cache* com replicação. O administrador do sistema deve ter em conta qual a configuração mais adequada para o caso. Existem dois mecanismos para repositórios de *cache*: *write-back*, as operações de escrita e leitura são feitas em *cache*, o cliente recebe imediatamente o resultado da operação, e eventualmente, os objetos migram por acção de um agente para o nível de armazenamento; *read-only*, as operações de leitura copiam objetos da camada de armazenamento para a camada de cache, no entanto, escritas são efetuadas diretamente para a camada

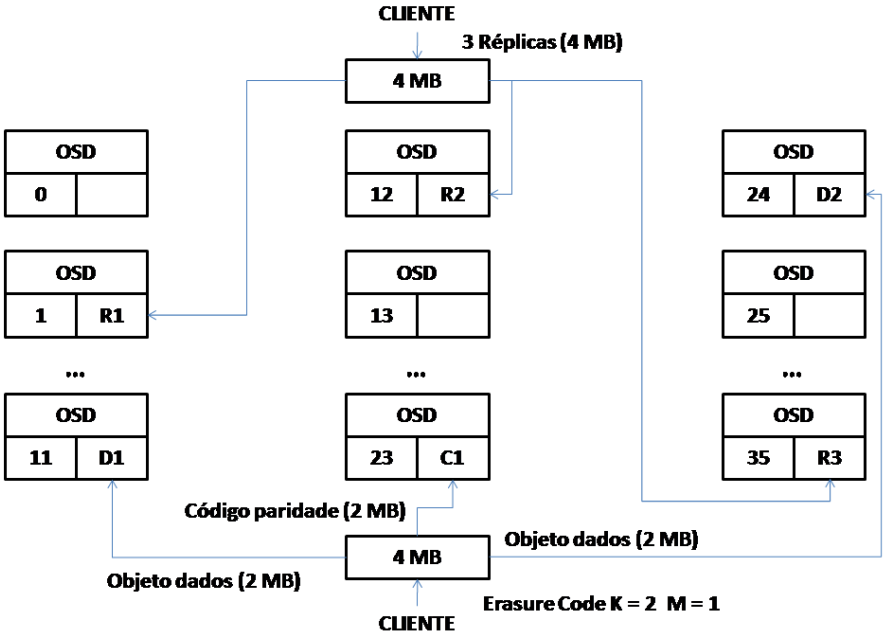


Figura 3.10: Comparação de uma *pool* replicada e *pool erasure coded*.

de armazenamento. Objetos obsoletos expiram e são removidos dos repositórios de *cache* com base na política definida. Um aspecto importante do mecanismo *read-only*, é apresentar consistência fraca pois o cliente pode ler versões de objetos (que estão em *cache*) e que no entanto não estão atualizadas. O objectivo de utilizar o mecanismo de cache como *read-only*, é fornecer armazenamento com desempenho melhorado, e em que o contexto não necessite de uma visão atualizada do objeto para que o sistema opere de uma forma correta, por norma neste contexto, os objetos não sofrem alterações em curtos intervalos de tempo (por exemplo a alteração de um template).

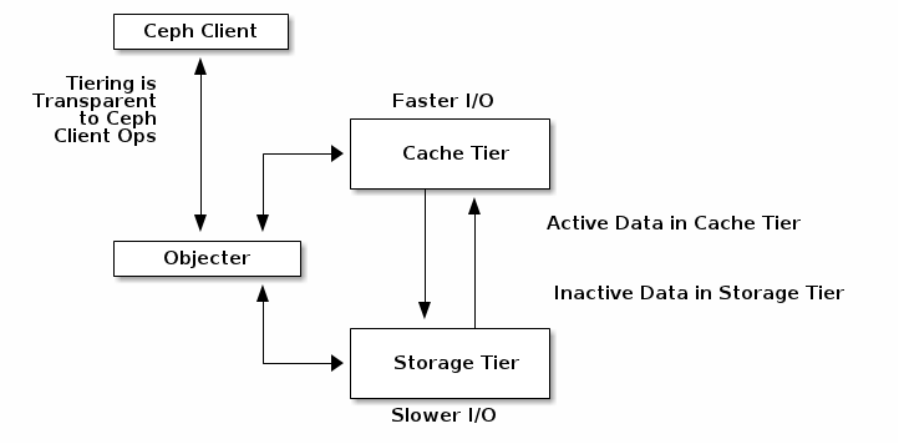


Figura 3.11: Camada de *cache* colocada "em cima" de uma camada de armazenamento.

Mapa do *Cluster*

Um servidor Ceph apenas necessita de saber a informação contida no mapa do *cluster*, de forma a poder realizar as diversas operações da sua responsabilidade. O mapa contém informações sobre monitores, OSDs, servidores de metadados, objetos e regras de posicionamento. O monitor principal é responsável por atualizar todas as informações do mapa, e replicá-las para outros monitores através de um algoritmo baseado no Paxos[Lam98; Lam01]. Os mapas são respetivos a uma época, isto é, existem mapas para várias épocas, é por considerado que tem várias versões do mapa; a existência de várias versões do mapa, permite aos OSDs, monitores e clientes obterem informações sobre o estado do Ceph e de todos os seus intervenientes não só na época atual, mas também nas anteriores.

O mapa de monitores, armazena dados cruciais para entrar em contato com monitores; tem identificador do *cluster* (*File System Identification* ou FSID) [1], a época do mapa, endereços e portas de monitor(es), e os dados da última alteração do mapa.

O mapa de OSDs, armazena informações para armazenamento de dados. Possui um FSID, a listagem de repositórios disponíveis, informações sobre a replicação e distribuição de dados em cada repositório, o estado de cada OSD, e datas de criação e modificação do mapa.

O mapa de PGs, armazena informações sobre grupos de colocação, ele armazena: a época do mapa, identificador do mapa de OSDs respectivo a essa época, o estado de cada grupo de colocação (por exemplo, pode ter menos do que o número desejado de réplicas para um determinado objeto) e outros dados de utilização estatística.

O mapa CRUSH, tem informação sobre distribuição e replicação de objetos, esta informação é partilhada por servidores e clientes, este mapa possui: uma lista de OSDs disponíveis; as regras de distribuição; e, a descrição hierárquica, que reflete uma suposta topologia do subsistema de armazenamento permitindo que os administradores de sistema possam estabelecer domínios de falhas e evitar eventuais gargalos existentes, é composto por *buckets* (nós abstratos, da árvore hierárquica CRUSH, de forma a mapear melhor os pontos de falha do domínio, os *buckets* podem-se referir ao servidor, à *rack*, ao bastidor, à sala, ao centro de dados, etc); e discos com o respectivo identificador, o peso, os algoritmos usados para *hashing* e para distribuição de dados. O administrador do sistema, pode alterar os valores de todas essas informações, para personalizar o mapa para o caso de uso.

O mapa MDS, contém informações específicas do servidores de metadados. Tem a

[1] É importante compreender que o *FSID* é um termo antigo, principalmente associado ao CephFS, quando o objetivo do Ceph era ser principalmente um sistema de ficheiros distribuído. A utilização FSIDs, permite que seja possível suportar diversos *clusters*, cada um com seu próprio identificador.

época, as datas de criação e modificação, informações sobre falhas anteriores, as configurações de ajuste e uma lista com informações sobre os servidores de metadados (endereço, porta, status, etc).

Visão Geral do CRUSH

A solução do Ceph para distribuição de objetos, não passa por tabelas pré-definidas, mas sim por uma função, CRUSH, que permite o seu calculo em memória. CRUSH, é um algoritmo determinista de distribuição de objetos, que revela ter no momento da distribuição uma natureza pseudo-aleatória. CRUSH é dito ser pseudo-aleatório, de forma que, em um conjunto de OSDs, consegue distribuir os objetos e as suas réplicas para OSDs aparentemente aleatórios (Figura 3.12 de [Wei06a]); no entanto, se o *cluster* não for modificado entre duas chamadas CRUSH, os destinos serão os mesmos, por isso é intitulado como sendo um algoritmo determinista.

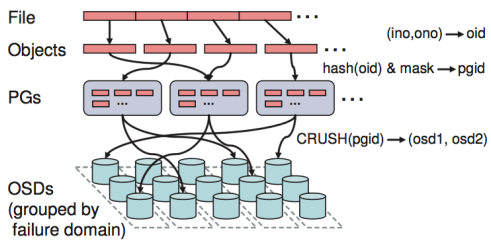


Figura 3.12: Ilustração de distribuição de dados utilizando o algoritmo CRUSH.

Para escrever um objecto, o cliente começa por contactar um monitor, para obter uma cópia atualizada do mapa do *cluster*, permitindo ao cliente calcular o OSD principal, que vai armazenar o objecto. O cliente, submete o identificador do objeto a uma função de *hashing*, que retorna o grupo de objetos correspondente, de seguida, submete o identificador do grupo à função CRUSH que retorna o conjunto de OSDs responsáveis por armazenar o objecto. O resultado da função de CRUSH, é um conjunto de OSDs activos, em que o primeiro é considerado o primário, e os restantes são utilizados para armazenar as réplicas. O cliente envia a operação de gravação para o primário, que, em seguida, redireciona a operação de escrita para as réplicas, espera que estas completem a operação de escrita, após o qual é a sua vez de submeter a escrita ao seu próprio armazenamento e responder no final ao cliente. A operação de leitura é semelhante, mas não existe qualquer interacção entre os OSDs primários e de réplica, em vez disso o principal lê o objeto e reconhece imediatamente para o cliente.

A configuração CRUSH é baseada em regras, o administrador do sistema é capaz de criar regras para armazenar objetos em subsistemas de armazenamento à sua escolha. Esta solução permite ter em conta a topologia física.

Descrição Detalhada do Algoritmo CRUSH

Cada regra consiste numa sequencia de operações aplicadas à hierárquia CRUSH (ver Algoritmo 1).

O valor de entrega da função CRUSH, x, tipicamente o identificador do grupo de objetos. O 1º procedimento, $take(a)$, seleciona um *item* (tipicamente um *bucket*) na hierárquia CRUSH, e associa-o ao vector $\overrightarrow{i}$, que serve *input* para as operações seguintes. O procedimento $select(n,t)$ itera sobre cada elemento do vector $\overrightarrow{i}$, $i \in \overrightarrow{i}$, e escolhe n *items* distintos do tipo *t* da sub-árvore do mesmo. Para cada $i \in \overrightarrow{i}$, o passo $select(n,t)$, itera sobre $r \in 1, ..., n$ *items* e desce recursivamente através de qualquer *bucket* intermediário, selecionando pseudo-aleatóriamente um *item* aninhado utilizando a função $c(r,x)$ (falamos sobre estas funções, no texto "Buckets e Dispositivos"), até que encontra o *item* de tipo t. Os n *items* encontrados são colocados novamente no vector $\overrightarrow{i}$, apartir daí ou são submetidos novamente ao procedimento $select(n,t)$, ou são colocados no vector resultado $\overrightarrow{o}$, com a operação $emit$.

Algorithm 1 CRUSH - colocação do objeto x

procedure TAKE(A) ▷ Colocar item *a* no vector $\in \overrightarrow{i}$
  $\overrightarrow{i} \leftarrow [a]$
end procedure
procedure SELECT(N,T) ▷ Selecionar *n* items de tipo *t*
  $\overrightarrow{o} \leftarrow \emptyset$ ▷ Resultado final, inicialmente vazio
  for $i \in \overrightarrow{\mathrm{i}}$ **do** ▷ Itera sobre $\overrightarrow{i}$
    $f \leftarrow 0$ ▷ Inicialmente não há falhas no item *i*
    for $r \leftarrow 1, n$ **do** ▷ Iterar sobre *n* réplicas
      $f_r \leftarrow 0$ ▷ Não há falhas na réplica
      $retry\_descent \leftarrow false$
      repeat ▷ Repetir até não ser possivel descer na sub-árvore
        $b \leftarrow bucket(i)$ ▷ Instanciar o *bucket* apartir do *item i*
        $retry\_bucket \leftarrow false$
        repeat ▷ Descer na árvore até encontrar *item* ou falhar 3 vezes
          if "first n" **then**
            $r' \leftarrow r + f$ ▷ Estratégia de agregar os 1os n *items* disponiveis
          else
            $r' \leftarrow r + f_r n$ ▷ Escolhe uma réplica *n* indices à frente para substituir esta réplica
          end if
          $o \leftarrow b.c(r', x)$ ▷ Encontrar pseudo-aleatóriamente um item (função selectora do *bucket b*).
          if $type(o) \neq t$ **then** ▷ Se não for o tipo *t*, então:
            $b \leftarrow bucket(o)$ ▷ Continuar a descer...
            $retry\_bucket \leftarrow true$ ▷ ...e procurar no novo *bucket*
          else if $o \in \overrightarrow{\mathrm{o}} \vee failed(o) \vee overload(o, x)$ **then** ▷ Há colisão (já foi escolhido), falhado ou sobrelotado
            $f_r \leftarrow f_r + 1, f \leftarrow f + 1$
            if $o \in \overrightarrow{\mathrm{o}} \wedge f_r < 3$ **then**
              $retry\_bucket \leftarrow true$ ▷ Procurar no mesmo *bucket* novos *items*
            else
              $retry\_descent \leftarrow true$ ▷ Escolher outro caminho apartir de *i*
            end if
          end if
        until $\neg retry\_bucket$
      until $\neg retry\_descent$
      $\overrightarrow{\mathrm{o}} \leftarrow [\overrightarrow{\mathrm{o}}, o]$ ▷ Adicionar ao output
    end for
  end for
  $\overrightarrow{\mathrm{i}} \leftarrow \overrightarrow{\mathrm{o}}$ ▷ Preparar *input* para uma eventual (nova) chamada select(n,t)
end procedure
procedure EMIT
  $\overrightarrow{\mathrm{R}} \leftarrow [\overrightarrow{\mathrm{R}}, \overrightarrow{\mathrm{i}}]$ ▷ Adicionar elementos encontrados ao resultado final
end procedure

O algoritmo CRUSH, é basicamente um conjunto de procedimentos que são definidos pelo administrador. A regra, por omissão, é a seguinte:

```
1     rule replicated_ruleset {
2     ruleset 0
3     type replicated
4     min_size 1
5     max_size 10
6     step take default
7     step chooseleaf firstn 0 type host
8     step emit
9     }
10
```

Figura 3.13: Regra CRUSH.

Um conjunto de regras (ou *ruleset*), estabelece: o identificador do conjunto de regras; o tipo de repositórios a que se destina (replicados ou *erasure coded*); número mínimo e máximo de réplicas que devem estar disponíveis; o passo para selecionar o *input* inicial, `step take default`; os passos necessários para descer na árvore de hierarquia CRUSH, `step chooseleaf firstn 0 type host` (no Algoritmo 1 equivalem ao procedimento `select(n,t)`); e finalmente, o passo `emit`, termina a o processo de escolha de OSDs. Por exemplo, seguindo Figura 3.14, podemos escrever regras para seleccionar um *bucket* chamado `room`, um bucket de tipo `row` , de seguida para selecionar 2 `buckets` de tipo `rack`, e em cada `rack`, selecione um `bucket` de tipo de `enclosure`, selecionando um dispositivo do `bucket`; assim, Ceph assegura que, mesmo se um bastidor falhar, uma réplica será armazenada num dispositivo, numa `rack` separada.

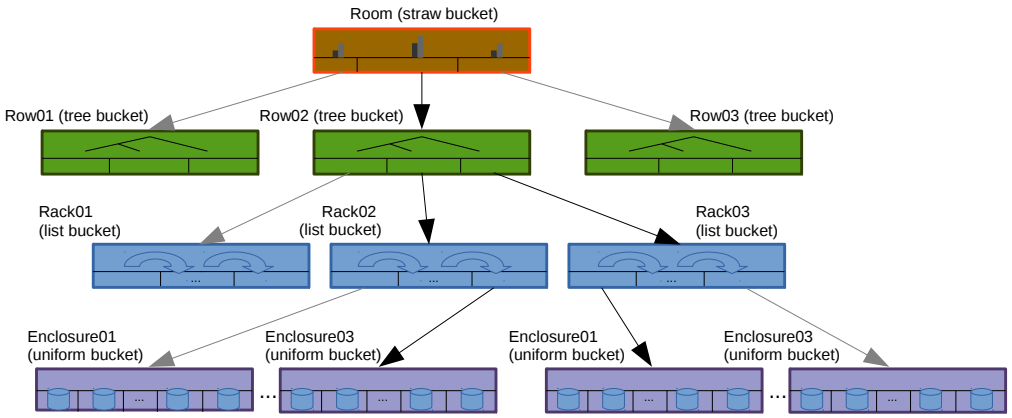


Figura 3.14: Exemplo de uma hierarquia CRUSH.

Buckets e Dispositivos

O CRUSH foi desenhado para fornecer eficiência e escalabilidade, enquanto tenta minimizar a migração de dados no momento que é necessário balancear o *cluster* após uma falha, adição ou remoção de um OSD. Por isso mesmo, existem vários tipos de *buckets*, cada *bucket* é baseado numa estrutura de dados interna; e cada tipo utiliza uma função pseudo-aleatória $c(r, x)$ para escolha de *items* internos, que representa uma troca entre complexidade computacional e eficiência reorganização de dados, complexidade no cálculo da localização do objeto em troca das operações de E/S necessárias para reequilibrar o *cluster*. A função tem como base no algoritmo de *hashing* Jenkins 32 bits [Jr.95] para escolha dos nós interiores, definido pela terminologia do Ceph como sendo *items*. O tipo de *buckets* existentes são: uniforme, árvore, lista e "palha" (ou *straw*).

O *bucket* uniforme, não utiliza os pesos dos dispositivos na sua lógica, por isso, os dispositivos têm igual probabilidade de ser escolhidos. Este tipo de *buckets* é utilizado para representar conjuntos de dispositivos com as mesmas características. É utilizado o princípio de *hashing* consistente, tendo como entrada o número de réplicas *r*, o identificador do grupo de colocação *x*, usando um número primo (*p*) escolhido aleatoriamente, e o número de *buckets*, m. A utilização de *hashing* consistente significa que a vantagem principal está na complexidade de cálculo da localização do objeto. Se há adição ou remoção de dispositivos, isto significa uma migração de dados para todos os dispositivos, esta solução não é a melhor em termos de organização dos *items*. A função utilizada em *buckets* uniformes é:

$$c(r, x) = (hash(x) + rp) mod(m)$$

Os *Buckets* do tipo lista, utilizam *linked lists* para guardar o seu conteúdo, e contêm *items* com peso. Por isso, como qualquer lista, não é indicado para ter grandes conjuntos de elementos, dado que tem de percorrer até *n* elementos até encontrar o elemento que quer. Utilizam um algoritmo derivado do RUSHp [ELM04] ; este algoritmo começa na cabeça da lista, no *item* adicionado mais recentemente, compara o resultado da função de *hashing* rjenkins, *hash(x, r, item)* (onde *x* é a identificação do objeto, *r* é a réplica, e o *item* é a identificação do *bucket* ou dispositivo), dependendo do resultado da função, ou o *item* corrente é escolhido ou o processo continua para os restantes elementos da lista. Este algoritmo apresenta um bom desempenho durante a adição de *items* (que são adicionados à cabeça da lista), no entanto, os *items* removidos a partir do meio e da cauda da lista resulta em movimento desnecessário, tornando este algoritmo adequado para *clusters* que apenas se expandem.

De seguida é relevante falar de *buckets* que utilizam árvores binárias (ver a Figura 3.15, retirada de [Wei06b]), com cada nó a saber o peso de suas sub-árvores da esquerda e da direita. Cada nó é marcado de acordo com a seguinte estratégia: a folha mais à esquerda na árvore é identificado com "1"; cada vez que a árvore é expandida, o novo nó passará a ser a nova raiz, sendo a raiz anterior o seu novo filho esquerdo, e o novo

nó de raiz passa a ter o rótulo da raiz anterior com um "0" como sufixo (por exemplo, 1, 10, 100, 1000); o filho direito tem o mesmo identificador como o mais à esquerda somado com o identificador do seu pai. A fim de escolher um *item* na árvore, CRUSH começa na raiz e calcula apartir da função de *hashing* $hash(x, r, b, item)$ (tomando como argumentos a identificação do objeto, número de réplicas, o identificador de *bucket*, e o rótulo do nó atual). O resultado da função, é comparada com a proporção em peso dos sub-árvores esquerda e direita. Se a árvore à esquerda tem mais peso, então ele vai ser escolhido, caso contrário ele irá escolher a árvore direita. *Buckets* que utilizam árvores binárias, são mais adequados a pesquisar quando há grandes conjuntos de nós aninhados, uma vantagem é a facilidade em adaptar-se à expansão e compressão do *cluster*.

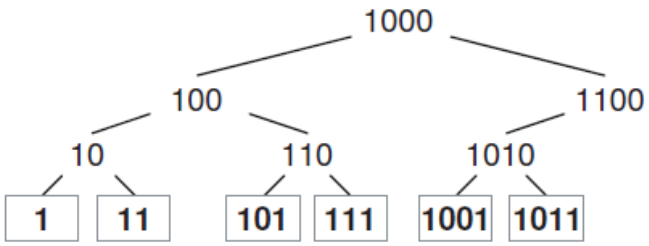


Figura 3.15: Exemplo de uma hierarquia CRUSH.

Os *Buckets* do tipo *straw*, utilizam o peso de cada *item* para distribuir uniformemente os objetos, tornando mais justa a escolha do *item* de destino, tal não ocorre com *buckets* de tipo lista e árvore). Para escolha da colocação do objeto, uma "palha" de tamanho aleatório é retirada por cada *item* no *bucket*, e o *item* com a "palha" mais longa é escolhido. O comprimento da "palha", é adquirida através da *hash* do grupo de colocação *x*, o número de réplicas *r* e o identificador do *item*, *i*; sendo o comprimento multiplicado pelo peso do *item* $f(w_i)$, para permitir que *items* com peso elevado possam competir de forma justa com *buckets* de menor peso, dado que estes *items* são capazes de armazenar mais objetos. A função de um *bucket* de tipo *straw* é:

$$c(r, x) = max_i(f(w_i)hash(x, r, i))$$

A escolha do tipo de *buckets* é baseada no tipo de subsistema que o administrador tem à sua disposição e a gestão que o subsistema irá ter (se irá ter eventuais expansões ou compressões). No caso em que é esperado termos um número fixo de *buckets*, então o melhor tipo é o uniforme; quando é esperado que os *buckets* continuem a expandir-se num ritmo acelerado, então é razoável utilizar *buckets* do tipo lista; em situações que é esperado adições e remoções constantes de *items*, então deve-se utilizar *buckets* do tipo *straw*; as árvores são soluções generalistas, em que prometem uma complexidade de pesquisa e organização de dados razoáveis.

Protocolo de Autenticação

A autenticação no Ceph é fornecida pelo protocolo de autenticação cephx. Em primeiro lugar, o administrador precisa criar um identificador para o utilizador e uma chave secreta, e armazená-lo num ficheiro *"keyring"*; esta chave deve ser por isso passada para o cliente de uma forma segura. No início de cada sessão, o cliente deve estabelecer um mecanismo que permite os servidores tomar imediatamente conhecimento que o cliente é quem ele diz ser; o cliente começa por requisitar uma chave de sessão para o monitor, em seguida, o monitor cria um desafio, ele cifra a chave de sessão com a chave secreta, que o cliente deve ser capaz de decifrar pois também tem a chave secreta; tendo a chave de sessão, o cliente deve executar a última etapa, que é, na verdade, dizer o monitor que resolveu o desafio, por isso, ele pede um bilhete, neste bilhete ele armazena a chave de sessão, que ele só tem porque ele podia decifrar a mensagem enviada pelo monitor. O bilhete será usado para atribuir a cada mensagem passada entre o cliente e os servidores. Este protocolo destina-se a ser usado entre clientes e servidores Ceph; quando se utiliza RADOS Gateway, o administrador deve criar utilizadores Swift/S3 para interagir com a interface RESTful Ceph.

3.4 Discussão

Os sistemas de ficheiros distribuídos apresentados neste capítulo fornecem migrações de máquinas virtuais em execução sem disrupção do serviço (que pode ser uma característica necessária no futuro) e para fornecer dados partilhados. De todos os sistemas de ficheiros apresentados, o menos adequado é o CephFS, pois não é adequado para um ambiente de produção, ainda; nestes casos, uma solução rápida é colocar uma camada de NFS, configurar quantos servidores NFS necessários, armazenamento de exportação de cada um para um conjunto específico de anfitrião, e configurá-lo para usar com Cinder. No que respeita à execução física, todos os sistemas de armazenamento objeto estudado (Ceph, Swift, Gluster) podem ser implementado no mesmo *hardware* de armazenamento *"off-the-shelf"*, desde existam otimizações. Por exemplo, Ceph, um caso de uso comum é usar uma camada de *cache* baseada em SSD, ou para armazenar o *journal* do sistema de ficheiros local em um disco SSD em separado, ou na ausência de discos SSD, usando diferentes partições para armazenamento de *journal* e dados discos mecânicos. GlusterFS é um sistema de ficheiros *shared nothing*, e requer mais *bus* de rede na camada da CPU, a fim de proporcionar um bom desempenho. O Sheepdog aparenta ser uma boa solução para núvens OpenStack, sendo a sua arquitetura linearmente escalável, dado que tem uma instalação distribuída, heterogénea ao nível da camada de armazenamento de objetos, e totalmente escalável pois a adição de novos nós implica uma direta melhoria de desempenho no acesso à camada de armazenamento. A grande vantagem face ao Ceph é ter uma estratégia de escrita direta, para disseminar as cópias dos objetos. No entanto, devido ao fato de as escritas em *journal* serem exclusivas para metadados, sendo a escrita

dos dados dos volumes direta para o disco, e tendo em conta que o Sheepdog é menos versátil em termos de funcionalidades, comparando este com o Ceph e o GlusterFS, isto levanta questões sobre a estabilidade deste sistema em escalar em grandes ambientes de produção.

Os sistemas de armazenamento mais promissores são: Ceph, GlusterFS, Cinder e Swift. Cinder e Swift, porque estes são os serviços nativos para OpenStack, isso significa que são naturalmente adequados para ambientes computação em nuvem; Ceph e GlusterFS, para que eles ofereçam uma solução de armazenamento unificada, bem como desempenho e vêm e tolerância de rede, o que significa que, mesmo que ocorram faltas, o tráfego é retomado de forma transparente. Embora o GlusterFS ofereça o mesmo acesso a dados como Ceph, o fato de que o que é um sistema de ficheiros distribuídos ao contrário Ceph (não CephFS), é uma desvantagem enorme, se o objetivo principal é fornecer armazenamento baseado em blocos e baseado em objetos. O Ceph tem como vantagem não ter o sistema de ficheiros distribuídos como seu núcleo, assim não ter que gerir metadados; além disso, tem a possibilidade de fornecer armazenamento compartilhado e migração em tempo real, integrado-se com NFS. Ceph revela-se o sistema de armazenamento mais adequado, uma vez que não é considerado um sistema de ficheiros distribuídos, mas que oferece os mesmos recursos que possam ser necessários; ele também oferece mais flexibilidade para gerir o *cluster*, com melhores controles de posicionamento e tolerância a faltas mais elevados do que os serviços nativos. As suas regras de distribuição, podem designar zonas específicas de falha, gargalos e outros problemas que possam surgir na infraestrutura.

No próximo capítulo, vamos intalar um *cluster* Ceph, experimentar várias configurações e avaliar cada uma.

4

Avaliação Experimental

A nuvem OpenStack a que se destina o Ceph, irá armazenar quantidades massivas de dados durante um período indeterminado, e irá suportar diferentes aplicações com comportamentos variáveis; será por isso, que o Ceph terá de se adaptar não só aos requisitos atuais, como o tipo de aplicações que a nuvem em produção já alberga, mas também a infraestrutura disponível e o orçamento para gestão; de futuro, a solução escolhida terá de conseguir-se adaptar a necessidades que possam surgir.

Este capítulo começa (na secção 4.1) por especificar a metodologia que utilizamos para o estudo do Ceph; a secção 4.2 descreve cada um dos *microbenchmarks* utilizados para obter os resultados apresentados neste capítulo; a secção 4.3 apresenta a caracterização da infraestrutura de testes, no qual o Ceph irá ser instalado e testado; na secção 4.4, apresentamos a configuração do Ceph; finalmente, na secção 4.5, apresentamos os resultados dos testes efetuados ao Ceph, e analisamos esses mesmos dados.

4.1 Metodologia

O estudo segue uma metodologia que pretende incluir várias características fundamentais do Ceph, tais como:

- Eficiência;
- Escalabilidade;
- Resiliência.

A avaliação tem por base a caracterização da infraestrutura, que serve como patamar de referência para posteriormente ser realizado comparações com o Ceph, são feitos testes com discos JBOD e RAID por hardware (maioritariamente RAID-0, mas também RAID-5, RAID-6 e RAID-10, em anexo). A eficiência do Ceph foi avaliada através de uma análise comparativa de desempenho com os testes com discos JBOD e RAID, no qual é esperado ver um melhor desempenho da solução hardware, dado tratar-se de uma solução de mais "baixo nível".

4.2 *Microbenchmarks*

4.2.1 Métricas

Várias métricas são utilizadas como meio de avaliação nas indústrias de armazenamento e de rede:

- Taxa de transferência de dados - Usualmente, é expressa em quantidade de dados que é possível ser entregue (num disco, adaptador de rede, etc), esta pode ser associada com transferências de um ficheiro, cujos blocos estão dispostos sequencialmente em disco. Taxas de transferência de dados são normalmente expressos em unidades por segundo, sendo neste trabalho apresentado em Gigabit por segundo (Gbps ou Gb/s) e em Megabyte por segundo (MBps ou MB/s). Por isso, seja para rede, seja para armazenamento, valores mais altos indicam melhor desempenho.
- Operações por segundo (IOPS) - Uma métrica relacionada com a quantidade de operações de escrita e leitura que é possível realizar por segundo. À semelhança da métrica anterior, valores mais altos representam melhor desempenho.
- Latência - Uma última métrica que tem alguma relevância para este estudo é a latência, o tempo necessário para uma operação completar; geralmente representado em milissegundos (ms), sendo que, valores mais baixos são representativos de melhores resultados. Existem diversos motivos para a latência ser afetada, sendo geralmente relacionados com a composição física da infraestrutura.

Para uma melhor compreensão de taxa de transferência de dados, IOPS e latência, vamos exemplificar como as métricas seriam aplicadas para medir o desempenho de um disco SAS. O disco é composto por pratos, ou discos magnéticos, que rodam a uma medida expressa por revoluções por minuto (RPM), este é o número de rotações que o prato irá realizar por minuto. Quando estudamos o desempenho de um disco, devemos ter em conta o tempo de acesso ao mesmo; este é medido em função do tempo de procura, tempo que demora ao braço a chegar à trilha certa, com a latência rotacional, tempo que demora ao disco magnético rodar para que o setor desejado esteja ao alcance da cabeça de leitura e escrita; isto, considerando que os dados que pretendemos ler estão armazenados sequencialmente, caso contrário o braço e o disco movimentam-se de novo, aumentando a latência.

É importante referir que existe uma relação entre o dimensão de dados a ler e escrever com IOPS; operações de leitura e escrita tendem a apresentar mais IOPS à medida que diminuímos a dimensão do bloco, ao aumentar o bloco essas mesmas operações tendem em apresentar menos IOPS, isto porque há um limite para a quantidade de operações que podemos fazer por segundo. Por exemplo, fazendo uma leitura sequencial aos discos SAS disponíveis, com blocos de 256 KB conseguimos 735 IOPS, com blocos de 16 MB conseguimos 11 IOPS; multiplicando a dimensão de bloco pelos IOPS conseguimos 188 MB/s e 176 MB/s, respetivamente, isto indica que quando os dados estão armazenados sequencialmente no disco é possível obter tempos semelhantes com dimensões de blocos diferentes. No entanto, na maioria dos casos os dados estão armazenados em setores aleatórios do disco, se realizarmos uma leitura aleatória com 256 KB e 16 MB, o que vemos é uma enorme disparidade entre os dois, 45 MB/s e 160 MB/s, com 177 IOPS e 10 IOPS. Isto leva-nos a outra questão muito importante, quantos dos dados requeridos estão alocados contiguamente, e quantos estão armazenados em setores aleatórios do disco.

Resumindo, embora a taxa de transferência seja um bom indicador para leitura/escrita de blocos de grande dimensão e sequenciais, o mesmo deixa de ser válido caso os blocos diminuem de dimensão e passem a ser aleatórios, para este caso irá haver mais latência devido ao tempo de acesso a disco e para isso é mais realista pensar em número de operações que vão ser executadas e a consequência que irá ter na latência.

4.2.2 Aplicações

Os resultados apresentados nesta dissertação, são conseguidos com auxilio a ferramentas de teste de redes, discos e sistemas de ficheiros; para a rede foi utilizada uma ferramenta chamada Iperf; para realizar os testes a nível de blocos foi utilizado uma ferramenta pouco conhecida chamada Flexibe IO (FIO); para realizar testes ao Ceph é utilizada a ferramenta interna do Ceph, chamado RADOS Bench.

4.2.2.1 Flexible IO

O Flexible IO (FIO), é uma ferramenta para avaliação de desempenho de sistemas de ficheiros e discos (acesso a blocos). Os testes FIO podem ser descritos num ficheiro no formato ini, podendo ser especificado o padrão de E/S, e o motor utilizado (`sync`, `libaio`, `posixaio`, etc), o número de *threads*, o comprimento da fila de pedidos, a dimensão do bloco, e a carga de trabalho total, entre outras especificações. No fim de cada teste é apresentado a descrição dos testes, os tempos de latência, a taxa de transferência, IOPS, informação sobre submissão e conclusão de operações de escrita e leitura segundo o comprimento da fila, tempos de execução, utilização do CPU e do disco.

Listing 4.1: Exemplo de um ficheiro de teste FIO.

```
1  [global]
2  ioengine=sync
3  filename=/dev/sdb
4  iodepth=1
5  rw=write
6  direct=1
7  size=4G
8  numjobs=1
9  [4K]
10 bs=4k
11 stonewall
```

global Demarca o inicio da configuração geral do teste.

ioengine Define como a submissão de operações E/S, vai ser feita (`sync`, `libaio`, etc).

filename Ficheiro a ser avaliado, e a ser a ser partilhado entre *threads*.

iodepth Comprimento da fila de espera de pedidos.

rw Tipo de operações de E/S a ser executado, podendo ser `write`, `read`, `randread`, `randwrite`, etc.

direct Igualado a 1, equivale a utilizar `O_DIRECT`, ou seja, não utiliza as páginas de *cache* do Linux.

size Carga de trabalho do teste.

numjobs Número de *threads* a executar por tarefa.

4k Nome da tarefa.

bs Dimensão do bloco para fazer escrita ou leitura, por omissão é 4 KB.

stonewall Informa todas as *threads* a ficarem em espera activa até todas as *threads* terminarem as suas tarefas até este ponto.

4.2.2.2 RADOS Bench

A ferramenta interna do Ceph, RADOS Bench, é atualmente a ferramenta mais utilizada pela comunidade para avaliar o desempenho interno do Ceph; permite criar e ler objetos de 4 MB (por omissão) e executar múltiplas *threads*. Retorna a taxa de transferência máxima, mínima e média juntamente com a latência, o desvio padrão associado e o número de operações efetuadas. Sendo uma boa ferramenta para avaliar a escalabilidade do Ceph. De seguida, é apresentado um exemplo de um comando RADOS Bench, para executar duas *threads* a lerem sequencialmente durante 60 segundos objetos de 4 MB:

```
$ rados bench 60 seq -b 4M -t 2
```

4.3 Ambiente de Testes

4.3.1 Caracterização Geral

A infraestrutura de armazenamento é composta por 3 *racks*, cada uma contendo: um servidor Dell PowerEdge R630, e três armários de discos com 12 discos Seagate Constellation ST4000NM0023 de 4 TB em cada um, perfazendo um total de 144 TB por *rack*. Os servidores têm quatro portas Ethernet, sendo duas a 1 Gb/s e outras duas a 10 Gb/s; todas estão ligadas, numa configuração tolerante a faltas, a dois *switches* Dell Force10, por um único caminho ativo (a outra porta é usada para tolerância a faltas). Cada servidor tem dois processadores Intel Xeon E5-2630 a 2.4 GHz e duas placas de memória DDR4 de 16 GB; tem ainda dois discos SAS (Serial Attached SCSI) de 300 GB e 10K rpm em RAID-1 onde está instalado o CentOS 6.

4.3.2 Desempenho da Infraestrutura de Rede a 10 Gb/s

Como referimos, do ponto de vista das ligações ativas a configuração de rede (Figura 4.1) é formada por ligações a 10 Gbps, que é a largura de banda mínima recomendada para uma infraestrutura deste tipo, já que, por exemplo, um único disco debitando a 175 MB/s saturaria imediatamente uma infraestrutura Gigabit Ethernet.

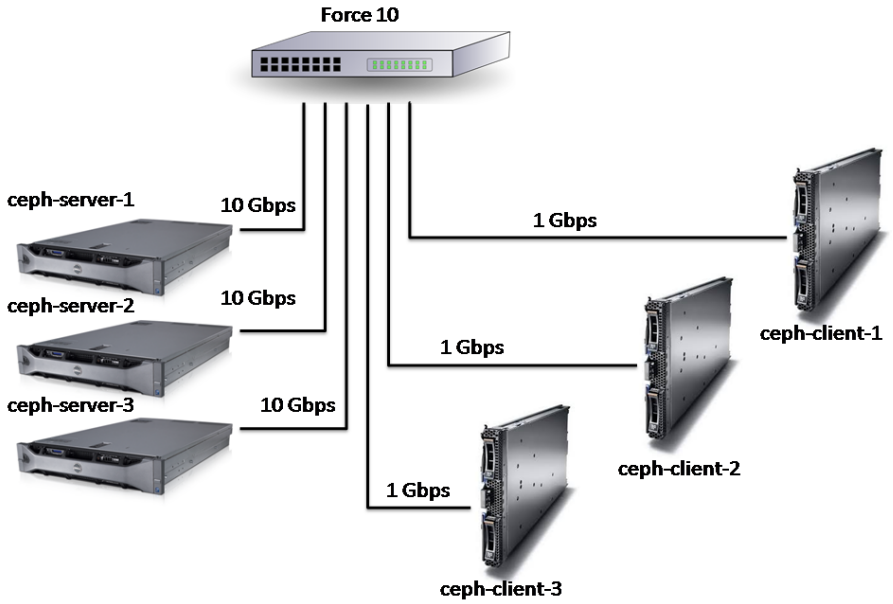


Figura 4.1: Rede da infraestrutura de testes.

Os testes focaram-se em determinar qual a maior largura de banda (LB) disponível a partir das conexões 10 Gb/s dos servidores PowerEdge R630, e verificar se o *switch* Force10 C300 seria capaz de suportar todo o tráfego gerado. Os testes foram realizados com o Iperf, uma ferramenta de testes de rede TCP/IP, e os resultados obtidos foram inesperados para quem está acostumado aos resultados em infraestruturas Gigabit Ethernet: as larguras de banda obtidas nas interfaces 10 Gb encontram-se muito próximos dos limites físicos da tecnologia, e não sofrem grande alteração quando se fazem variar parâmetros que fazem enorme diferença na tecnologia Gigabit, tais como o valor do MTU, (tabela 4.1) ou a dimensão das mensagens (tabela 4.2). A utilização de *full-duplex*, i.e., de dois *streams*, um envio e outro de receção, sobre a mesma interface não provocam qualquer degradação (tabela 4.3), enquanto a utilização de dois fluxos de envio sobre a mesma interface resulta em metade da largura de banda, como seria de esperar (tabela 4.4).

Para a determinação da LB em função do MTU utilizaram-se dois servidores, sendo um configurado como emissor (cliente iperf) e outro como recetor (servidor iperf); a execução do cliente iperf foi:

```
$ iperf -c <ip cliente> -M <mss> -m -t 600
```

Para os valores de `<mss>` usamos 1460 e 7960, para MTUs de 1500 e 8000, respetivamente. Estes valores obtêm-se descontando o *overhead* do TCP e do IP, que perfaz 40

bytes.

MTU 1500	MTU 8000
9.34 Gbps	9.85 Gbps

Tabela 4.1: Larguras de Banda para o MTU *standard* (1500) e *jumbo* (8000).

Os mesmos servidores foram usados para a determinação da LB em função da dimensão da mensagem (usando um MTU de 8000), usando o comando:

```
$ iperf -c <ip cliente> -l <dimensao do buffer>
```

Tamanho de Buffer						
4 KB	16 KB	64 KB	256 KB	1024 KB	4096 KB	16384 KB
9.89 Gbps	9.89 Gbps	9.89 Gbps	9.89 Gbps	9.89 Gbps	9.89 Gbps	9.89 Gbps

Tabela 4.2: Larguras de Banda em função da dimensão da mensagem, em *half-duplex*.

A avaliação da LB em *full-duplex* e em função da dimensão da mensagem (usando um MTU de 8000), fez-se usando mesmos servidores e com o comando:

```
$ iperf -c <ip cliente> -l <dimensao buffer> -d
```

Tamanho de Buffer	Nó 1	Nó 2
4 KB	9.88 Gbps	9.86 Gbps
16 KB	8.88 Gbps	9.86 Gbps
64 KB	9.87 Gbps	9.86 Gbps
256 KB	9.81 Gbps	9.81 Gbps
1024 KB	9.86 Gbps	9.86 Gbps
4096 KB	9.86 Gbps	9.86 Gbps
16384 KB	9.86 Gbps	9.86 Gbps

Tabela 4.3: Largura de Banda em função da dimensão da mensagem, em full-duplex.

A última avaliação foi a da LB usando dois fluxos de envio *half-duplex* e fez-se também em função da dimensão da mensagem (usando um MTU de 8000); usaram-se os mesmos servidores e o comando:

```
$ iperf -c <ip cliente> -l <dimensao buffer> -P 2
```

A conclusão é a de que esta infraestrutura de 10 Gb vale bem o seu custo (mais de 500 euros por porta) pois a LB que se obtém é muito próxima do máximo que o nível físico da tecnologia permite, seja qual for a situação: fluxos a half ou a full-duplex, MTU standard ou Jumbo, pacotes de média (4 KB) a muito grande (16 MB) dimensão.

Tamanho de Buffer	Stream 1	Stream 2
4 KB	4.94 Gbps	4.94 Gbps
16 KB	4.94 Gbps	4.94 Gbps
64 KB	4.87 Gbps	4.88 Gbps
256 KB	4.94 Gbps	4.94 Gbps
1024 KB	4.94 Gbps	4.94 Gbps
4096 KB	4.94 Gbps	4.94 Gbps
16384 KB	4.94 Gbps	4.94 Gbps

Tabela 4.4: Largura de Banda em função da dimensão da mensagem, para dois fluxos de envio.

4.3.3 Descrição do Subsistema de Armazenamento

Associado a cada nó (servidor), existe um subsistema de armazenamento composto por três armários (enclosures) de discos, interligados em cadeia (*daisy-chain*) entre si e ao adaptador instalado no servidor; os armários são Dell DAS MD1400, o adaptador é um Dell PERC HD830 e o servidor um Dell PowerEdge 630R. Cada armário suporta um máximo de 12 discos, e os discos utilizados na infraestrutura são discos SAS ST4000NM0023 (Serial Attached SCSI) magnéticos de 7200 rpm e 4TB.

Os valores relevantes, anunciados pelos fabricantes, para os desempenhos dos componentes e suas interligações são:

- Disco:
  - acesso sequencial: 175 MB/s
  - *cache*: 128 MB
  - interligação disco/armário: SAS a 6 Gb/s
- Armário:
  - 4 portas SAS a 12 Gb/s
- Adaptador (interligação servidor/armários):
  - 2 portas SAS a 12 Gb/s
  - *cache*: 2 GB

As interligações, exibidas na Figura 4.2, mostram um subsistema com redundância entre o adaptador e armários, formando um único ”caminho” de dados ativo a 12 Gb/s (por exemplo, podemos considerar que o caminho a azul é o ativo, sendo o de cor laranja usado caso haja uma falta no primeiro).

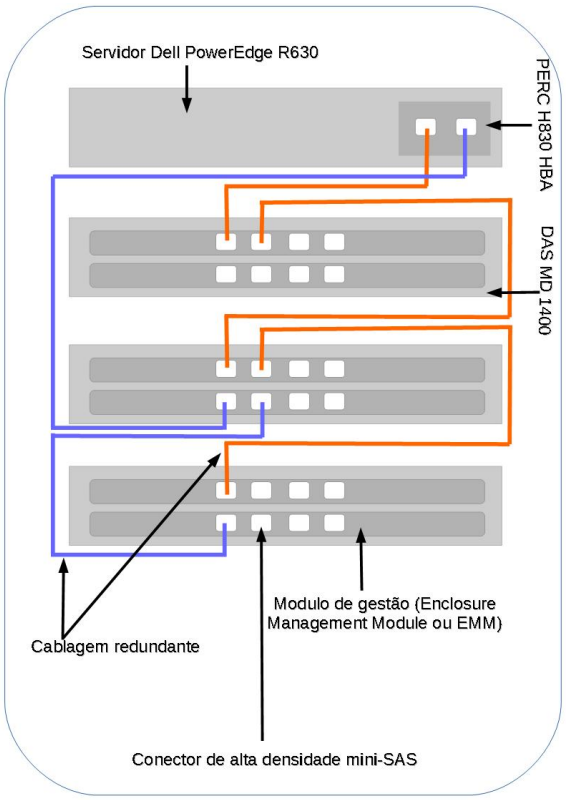


Figura 4.2: Interligação de um servidor PowerEdge R630 a três armários PowerVault.

4.3.4 Desempenho do Subsistema de Armazenamento

O desempenho do subsistema de armazenamento foi avaliado (através do FIO) tendo em conta o seu propósito que, genericamente, é o de albergar máquinas virtuais numa infraestrutura OpenStack; em particular, os discos presentes no subsistema vão constituir uma ou mais *pools* de armazenamento para o Ceph, um sistema de armazenamento baseado em objetos (ou *object-based storage*). As VMs, por sua vez, podem subdividir-se em dois grupos: o de imagens, ou *templates*, que são objetos maioritariamente de leitura a partir dos quais se criam, por clonagem, as instâncias (as VMs) que são depois executadas, e que constituem o segundo grupo de objetos, estes agora com um padrão variado de leituras e escritas.

Há assim três fases distintas no acesso aos objetos que representam as instâncias: fase de arranque (*boot*), em que o objeto-instância é essencialmente lido pelo hipervisor para criar a imagem em memória da VM; fase de operação, ou produção, na qual a VM gera um tráfego de leitura/escrita sobre o objeto que depende em muito da aplicação que esta executa; e fase de *shutdown*, em que o tráfego é geralmente de escrita, e relativamente pouco.

Em face do anteriormente exposto decidimos efetuar três análises de desempenho distintas ao subsistema de armazenamento: na primeira, avaliamos as suas capacidades "brutas" (raw) medindo as Larguras de Banda (LB) e número de operações de E/S (IOPS) em função do número de discos, da sua distribuição pelos armários, e do seu agrupamento em volumes RAID-0; na segunda, avaliamos os mesmos parâmetros (LB e IOPS) para várias configurações do Ceph - função do número de servidores de objetos (OSDs) e do grau de redundância/replicação desejado; finalmente, avaliamos os tempos de criação (clonagem), arranque e *shutdown* de VMs.

4.3.4.1 Desempenho "bruto" de uma Unidade do Subsistema de Armazenamento

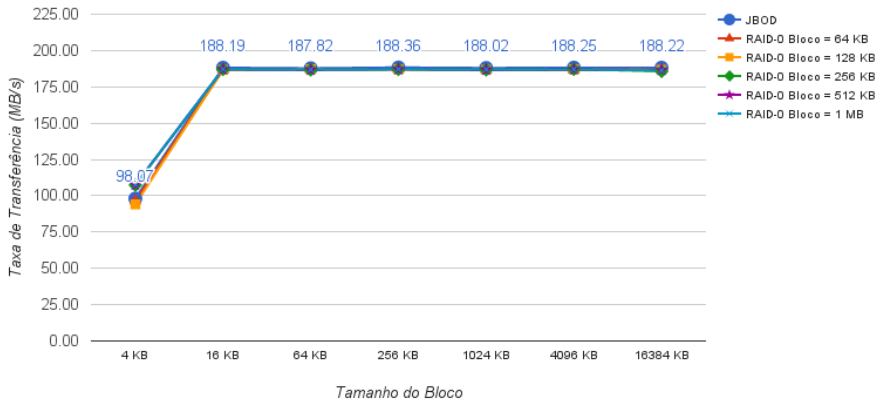
Os testes de desempenho aos discos físicos - acesso ao nível do *block device* - em "*raw*", ou seja, não-formatados, destinam-se a avaliar os limites de cada componente: discos, armários e interconexões (disco/adaptador do armário e armário/adaptador do servidor). Como estamos em presença de interligações não-partilhadas, i.e., cada armário está ligado a um único servidor, não nos interessa usar os mecanismos de redundância oferecidos pelos adaptadores RAID, deixando essa tarefa para o Ceph; assim, testamos unicamente agrupamentos de discos em RAID-0, aproveitando o aumento de largura de banda oferecido por este nível RAID.

Os testes destinam-se a determinar as duas métricas fundamentais para um sistema de armazenamento: a sua Largura de Banda (LB) e o número de operações que é capaz de executar por segundo (IOPS), tanto em leitura, como em escrita. Como ambas dependem da dimensão do *buffer* usado na operação, e como a dimensão por omissão de um "segmento Ceph" é de 4 MB, realizamos testes para dimensões de *buffer* compreendidas entre 4 KB (dimensão típica de um acesso a um sistema de ficheiros "instalado" num sistema

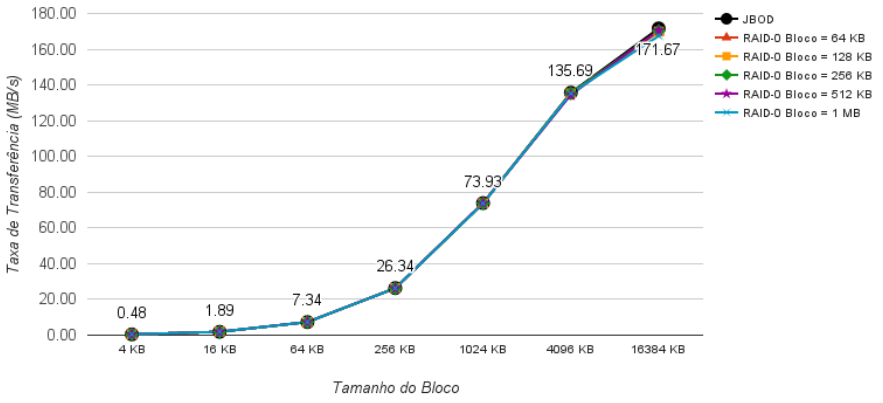
de operação com páginas de 4 KB) e 16 MB.

Desempenho de uma Unidade de Disco

As Figuras 4.3 e 4.4 mostram os valores obtidos para a LB e IOPS em leitura e escrita; os testes foram executados usando operações de leitura e escrita síncronas, e que não usam a *cache* do sistema de operação, o que se consegue usando a opção `O_DIRECT` no `open()` do "*raw device*", nem a *cache* do adaptador PERC H830, no qual foi também desativado o mecanismo de *read-ahead*. São apresentadas várias curvas de desempenho, com etiquetas tais como "JBOD" e "RAID-0 Bloco = ..."; tais etiquetas são usadas para descrever a configuração adaptador/disco. Assim, a etiqueta JBOD é usada para assinalar uma configuração em que o adaptador acede ao disco da forma mais "pura", e com um mínimo de *overhead*, enquanto que as etiquetas do tipo "RAID-0 Bloco = ..." assinalam que o disco se comporta como um disco constituinte de um grupo RAID-0, sendo em cada acesso movimentados o número de blocos necessários para perfazer a dimensão especificada na etiqueta.



(a) Leitura sequencial.

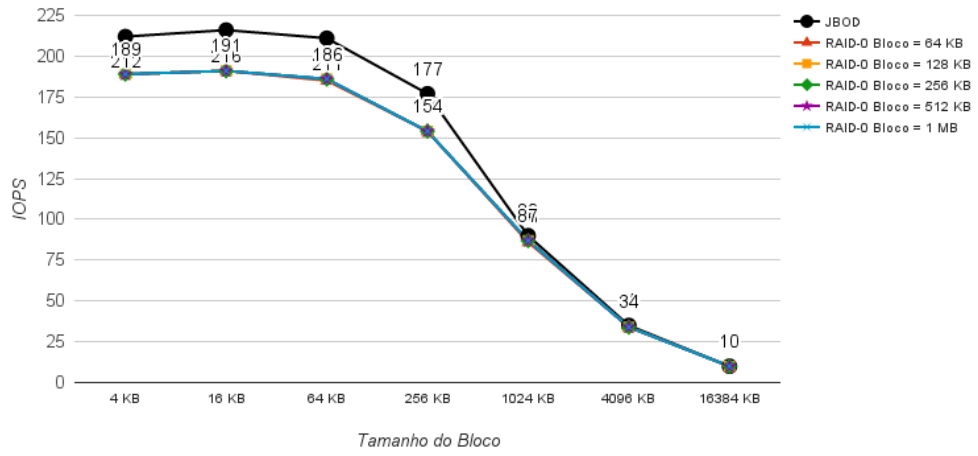


(b) Escrita sequencial.

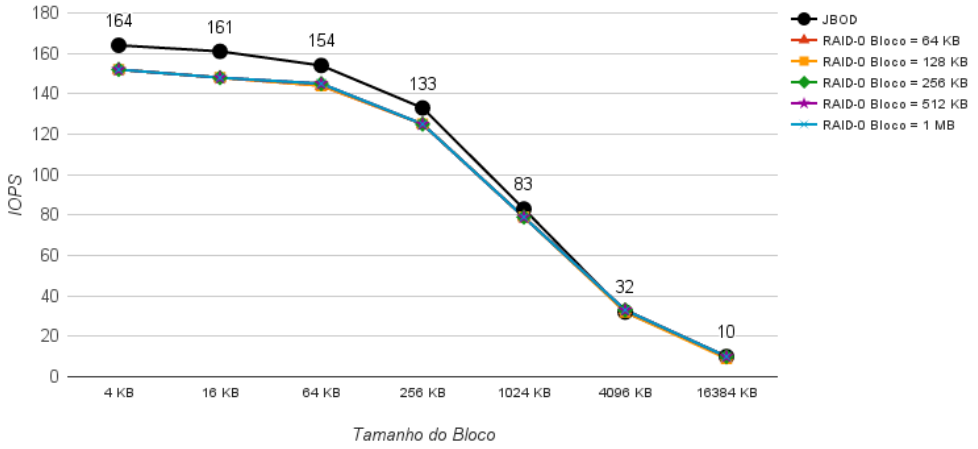
Figura 4.3: Largura de banda para um disco isolado, e um volume RAID-0 formado por um único disco, e com uma dada *stripe size*.

Em resumo, a LB em leitura sequencial começa com um valor de 98 MB/s, muito elevado para um *buffer* de 4 KB, e cresce progressivamente até atingir os 188 MB/s, um pouco acima do valor máximo especificado pelo fabricante que, recorde-se, era de 175 MB/s. Para a escrita sequencial o comportamento é idêntico, sendo que o valor máximo atingido é de 170 MB/s. Note-se ainda que a interposição do mecanismo RAID tem um *overhead* muito pequeno. É possível que o valor anormalmente elevado encontrado na leitura se deva à *cache* interna do disco (16 MB) e a um mecanismo de *read-ahead*, também interno à unidade, e que é ativado ao ser observado o padrão sequencial de acessos em leitura; e, como a escrita foi especificada como *write-through*, o efeito da *cache* interna é, nesse caso, nulo.

O teste de desempenho em acesso aleatório permite obter o valor de operações por segundo; a Figura 4.4 mostra o desempenho de um disco em leituras e escritas aleatórias.



(a) Leitura aleatória.



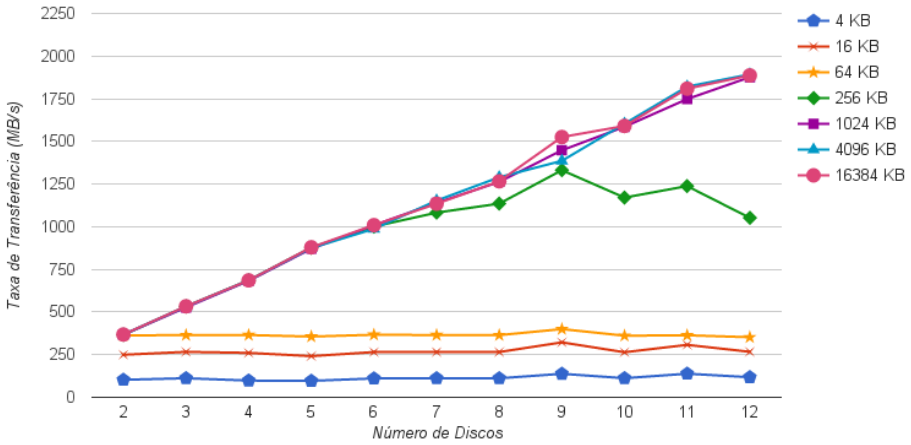
(b) Escrita aleatória.

Figura 4.4: IOPS para um disco isolado, e um volume RAID-0 formado por um único disco, e com uma dada *stripe size*.

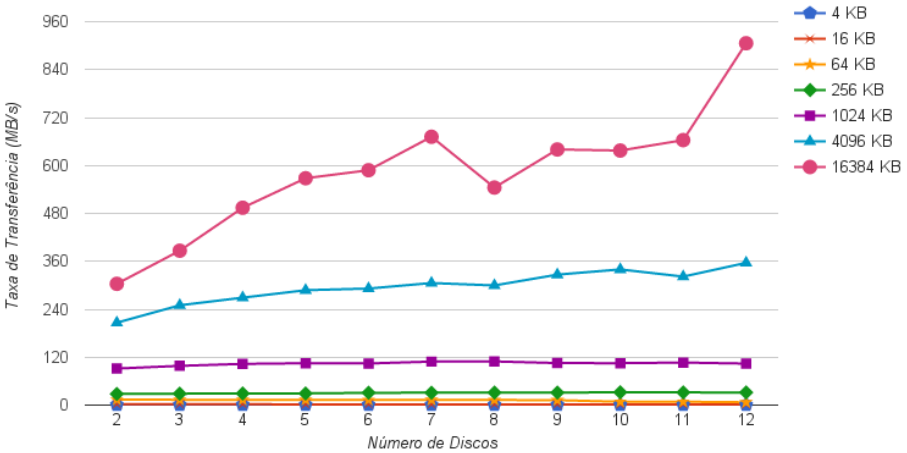
Os testes de desempenho em acesso aleatório, atingiram um máximo de 216 IOPS para leituras com acessos de 16 KB, e um máximo de 164 IOPS para escritas com *buffer* de 4 KB, ambos obtidos com a configuração JBOD. Novamente se nota que o *overhead* do RAID é relativamente pequeno.

Desempenho com Múltiplas Unidades de Disco num Armário

Para avaliar a forma como evolui o desempenho do subsistema à medida que se adicionam discos, começamos por avaliar o desempenho adicionando sucessivamente discos a um único armário, e formando com estes um único volume RAID-0, tendo-se ensaiado com diferentes valores para a dimensão do *buffer* de acesso. A Figura 4.5 mostra as larguras de banda assim obtidas tanto em leitura como em escrita sequencial.



(a) Largura de Banda em leitura.



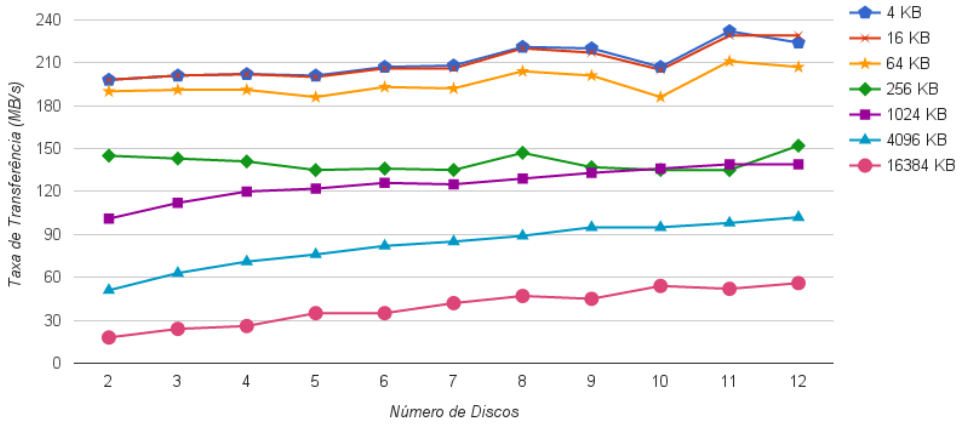
(b) Largura de Banda em escrita.

Figura 4.5: Evolução da LB num volume RAID formado por um número crescente de discos colocados num único armário.

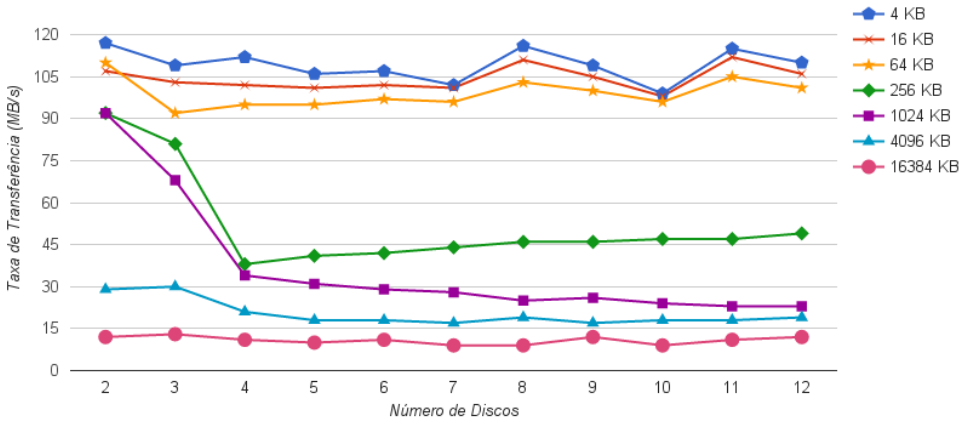
Os resultados obtidos mostram um efeito muito positivo do aumento do número de discos na evolução da LB, para acessos em leitura; todavia os resultados são dececionantes ao nível da escrita, indicando claramente a necessidade de se dispor de *caching* tanto

ao nível do adaptador RAID como ao nível do sistema de operação e sistema de ficheiros do servidor.

De seguida, mostramos os resultados conseguidos ao nível dos IOPS quando incrementamos o número de discos que constituem um volume RAID-0, tendo colocado todos os discos num único armário e feito variar a dimensão do *buffer* de leitura ou escrita aleatória.



(a) IOPS em leitura.



(b) IOPS em escrita.

Figura 4.6: Evolução do número de IOPS num volume RAID formado por um número crescente de discos colocados num único armário.

Os testes de IOPS com operações de leitura aleatória apresentam valores elevados, quando comparado com operações de escrita aleatória; novamente, os IOPS diminuem à medida que o tamanho do bloco aumenta. O volume RAID-0, com dois discos atinge 199 IOPS com um tamanho de bloco do RAID, com operações de 4 KB. Os IOPS tendem em aumentar ligeiramente com um maior número de discos no volume RAID-0, chegando a 232 IOPS com operações de leitura de 4 KB com 11 discos. Aumentando a dimensão do bloco a ser lido, permite obter 18 IOPS para dois discos.

Os testes de IOPS com operações de escrita aleatória apresentam o resultado mais elevado, 122 IOPS, com um volume RAID-0 com dois discos; aumentando a dimensão do bloco, resulta em redução de IOPS, conseguindo-se para um bloco de 16 MB, 12 IOPS. Acrescentando mais discos ao volume RAID cria oscilações nos resultados; tomando como exemplo um volume RAID-0 com 24 discos, operações de 4 KB o resultado é de 107 IOPS, enquanto que para escritas de blocos de 16 MB, obtém-se 9 IOPS. No entanto, podemos concluir que para a maioria dos casos os IOPS mantêm um valor semelhante independentemente do número de discos no volume.

Desempenho com Múltiplas Unidades de Disco Distribuídas por Dois Armários

Este grupo de testes foi realizado de forma muito simples: usamos um número crescente, par, de discos distribuídos igualmente por dois armários, e repetimos os mesmos testes executados na secção "Desempenho com Múltiplas Unidades de Disco num Armário". Os resultados obtidos podem ver-se nas Figura 4.7, para a avaliação da Largura de Banda, e na Figura 4.8, para a avaliação do número de IOPS. Nestes testes, a dimensão da *stripe* é de 64KB.

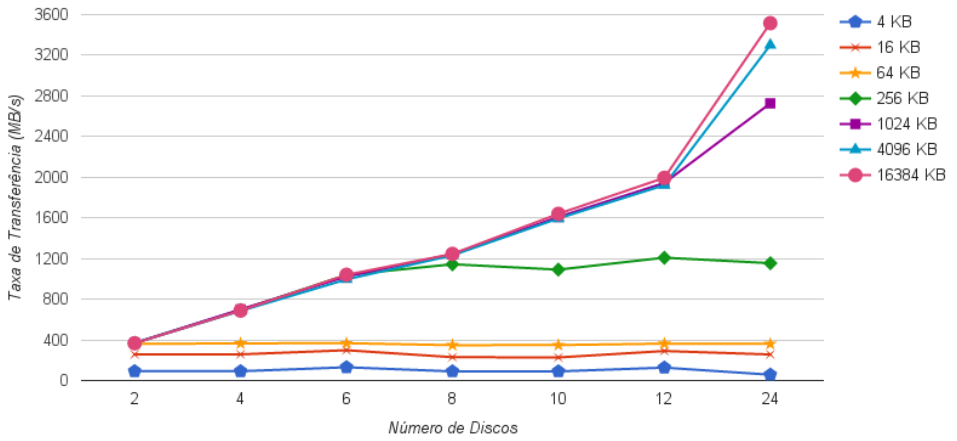
Quando comparamos os resultados obtidos na LB usando dois armários com idênticas configurações de discos em um único armário, notamos que as diferenças não são muito relevantes, com exceção de um valor anómalo, que não pudemos verificar posteriormente, para a LB da escrita com um RAID-0 de 12 discos, que num só armário apresentou um valor muito superior ao conseguido com os discos distribuídos por dois armários (900 MB/s contra 640 MB/s).

Por último, foram realizados testes de desempenho com discos de dois armários, e foram obtidos os IOPS presentes na Figura 4.8.

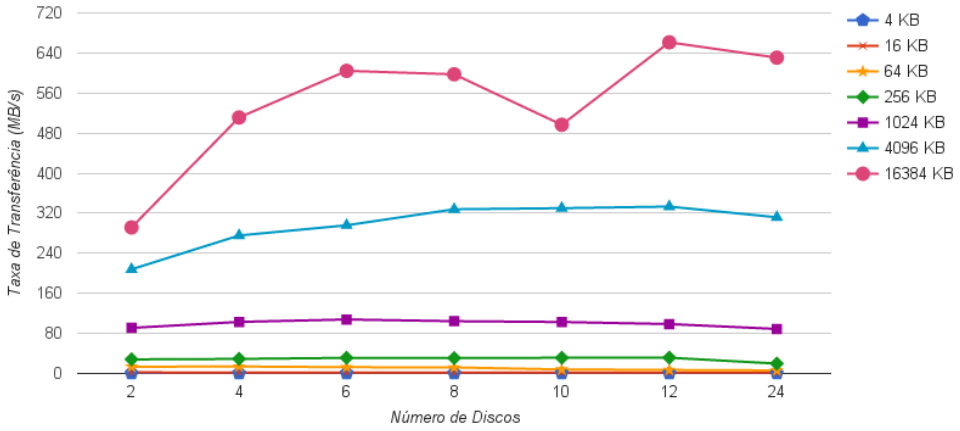
No caso do desempenho ao nível dos IOPS nota-se uma pequena vantagem da configuração que usa discos distribuídos pelos dois armários no caso do número de discos ser pequeno; quando este se aproxima da capacidade máxima do armário, as diferenças esbatem-se.

4.4 Configuração do *Cluster*

A instalação do *cluster* Ceph, refere-se à versão 0.80, intitulada Firefly. A instalação é auxiliada por uma ferramenta da administração, `ceph-deploy`, instalada no primeiro nó (`ceph-server-1`), esta ferramenta está incluída nos pacotes de instalação, e tem como objetivo principal a automatização de alguns processos e instalação; ainda no mesmo nó foi instalado um monitor (`mon.0`), e 12 OSD (`osd.0` até `osd.11`); no segundo nó (`ceph-server-2`) foram instalados mais 12 OSDs (`osd.12` até `osd.23`); finalmente, no terceiro nó (`ceph-server-3`) foram instalados os últimos OSDs do grupo (`osd.12` até `osd.23`); resumindo, temos 1 monitor e 36 OSDs.

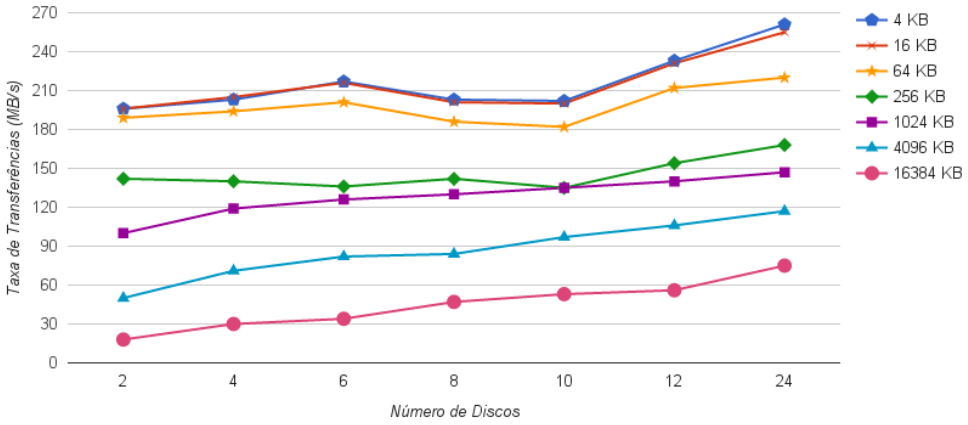


(a) Largura de Banda em leitura.

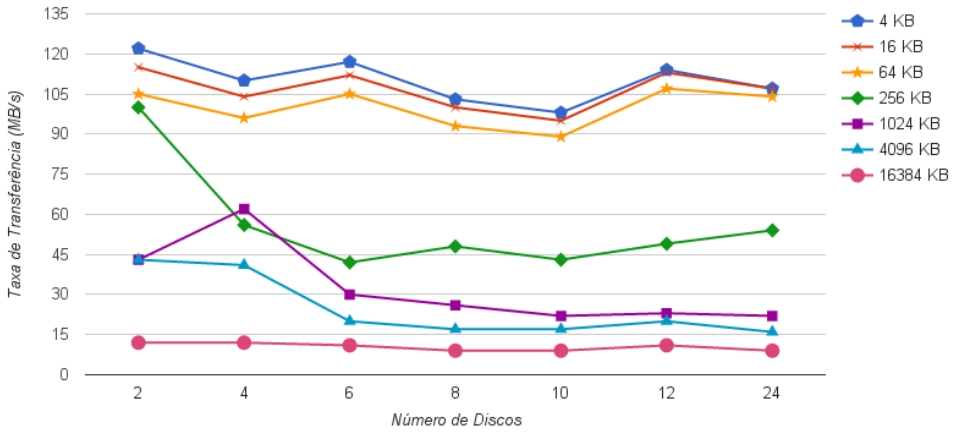


(b) Largura de Banda em escrita.

Figura 4.7: Evolução da LB num volume RAID formado por um número par crescente de discos equitativamente distribuídos por dois armários.



(a) Leitura aleatória.



(b) Escrita aleatória.

Figura 4.8: Evolução da IOPS num volume RAID formado por um número crescente de discos distribuídos por dois armários.

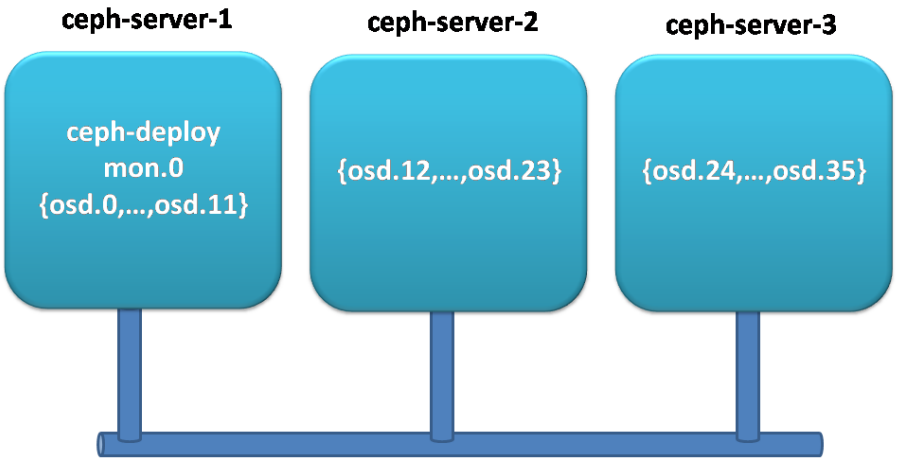


Figura 4.9: Distribuição de monitor e OSDs pelas máquinas.

A instalação é feita por uma série de comandos dados ao `ceph-deploy` com recurso a dois ficheiros de configuração chamados `ceph.conf` e `crushmap`.

O ficheiro de configuração ceph.conf é constituído por secções, no exemplo podemos ver que temos uma secção `[global]`, esta secção destina-se a configurações mais abrangentes; é ainda possível colocar uma secção dedicada a monitores (`[mon]`), uma secção dedicada a OSDs (`[osd]`); uma configuração mais especifica, desde que referido o identificador do OSD, por exemplo para um OSD com o identificador 0 (`[osd.0]`). O ficheiro `ceph.conf` necessita de um conjunto mínimo de configurações sem o qual o *cluster* não consegue atingir um estado operacional. Caso o número de nós seja inferior a três, é necessário definir o número de instâncias de um objeto, ou seja, o objeto e a(s) sua(s) réplica(s), com um valor inferior a três e superior a 1; isto deve-se ao facto de que (por omissão) estar definido que há sempre 3 instâncias de um objeto. Para definir o número de instâncias de um objeto deve-se incluir a variável `osd_pool_default_size` no ficheiro `ceph.conf`. Na criação do *cluster* é necessário definir o(s) sistema(s) anfitrião(ões) para a instalação do(s) monitor(es); deve ser especificado no ficheiro de configuração, nas variáveis `mon_addr` e `mon_host`, com o(s) IP(s) e *hostname(s)*, respetivamente. O identificador do *cluster* é definido através da variável `fsid`, a utilização de um identificador permite reutilizar os mesmos processos monitores, OSDs para múltiplos *clusters*. Por último, é necessário especificar a rede para o tráfego do *cluster*, isto define-se na variável `public_network`. Para além de ser definido as variáveis necessárias para obter um *cluster* operacional, é possível definir as regras CRUSH a serem escolhidas (em caso de ser omitido no comando) no momento de criação de *pools*, através da variável `osd_pool_default_crush_replicated_ruleset`. Na listagem 4.2 é ainda possível observar que foram desativadas as autenticações de modo simplificar a gestão do sistema que serve apenas para avaliação; `auth_service_required`, refere-se a autenticação de servidores do Ceph perante o contacto com clientes; `auth_client_required`,

estando habilitada, os clientes necessitam de se autenticar no momento que contactam um servidor; `auth_cluster_required`, refere-se a autenticação entre qualquer servidor do Ceph (OSD, monitor ou MDS).

Listing 4.2: Exemplo configuração do *cluster* (ficheiro *ceph.conf*).

```
[global]
  fsid = 2a22b7a8-de63-4606-9060-269885c84318
  mon_host = ceph-server-1
  mon_addr = 10.193.3.20:6789
  osd_pool_default_size = 1
  public_network = 10.193.0.0/16
  osd_pool_default_crush_replicated_ruleset = 0
  auth_service_required = none
  auth_client_required = none
  auth_cluster_required = none
```

O monitor é um processo leve que mantém uma cópia do mapa do *cluster*, e é responsável pela atualização e distribuição da mesma pelo outros intervenientes do sistema. Para uma descrição mais detalhada dos monitores, consultar o capitulo anterior, secção 3.3.6. A instalação de apenas um monitor, quando podemos formar um quórum deve-se ao facto de um monitor ser mais do que suficiente para a gestão do número de 36 OSDs; um monitor é mais do que suficiente para esta configuração pois este apenas apresenta alguma atividade no momento em há modificação de estado de algum interveniente do sistema, seja quando um OSD, monitor ou MDS é adicionado, seja quando um destes fica incontactável, nesse caso é enviado uma cópia atualizada do mapa do *cluster* a quem o requisitar (os monitores não fazem difusão das mudanças no *cluster*, em vez disso apenas informam os intervenientes que necessitam de uma cópia atualizada do *cluster*). No entanto, para ambientes de produção é recomendado a instalação de mais de dois monitores,de modo a tolerar faltas. Num ambiente de produção recomenda-se ainda, que o número de monitores seja ímpar, pois é obrigatório que uma maioria de monitores esteja contactável de forma a formar um consenso, sobre a alteração do mapa do *cluster*. Por isso, para uma instalação de dois monitores, não há tolerância a faltas; numa instalação de três monitores, apenas há tolerância a falta de um; para a instalação de quatro monitores, apenas há tolerância a falta de um; para uma instalação de cinco monitores, há tolerância a duas faltas; o mesmo se aplica para instalações com mais de cinco monitores.

Relativamente à instalação de OSDs, é feita tendo em conta que apenas temos discos SAS 6 GB, ou seja, temos um conjunto de discos, todos com os mesmos especificações. A cada OSD é dedicado um disco, tendo sido criado uma partição para o *journal*[1] e uma partição para o XFS. Limitamos o número de discos consumidos a 12 discos por servidor (consumindo uma caixa de discos por servidor), totalizando 36 discos. No final da instalação, podemos fazer uma consulta da hierárquia CRUSH (listagem 4.3), através do

[1]Um *journal* é um ficheiro que guarda todas as mudanças a serem feitas no sistema de ficheiros, isto antes de escrever os dados no disco por via de sincronização.

comando ceph osd tree, este resultado permite concluir que todos os OSDs estão ativos e dentro do *cluster*.

Listing 4.3: Exemplo comando *ceph osd tree*.

```
# id  weight  type name up/down reweight
-1  131 root storage
    -2  43.68   host ceph-server-1
         0   3.64     osd.0   up  1
         1   3.64     osd.1   up  1
         2   3.64     osd.2   up  1
         3   3.64     osd.3   up  1
         4   3.64     osd.4   up  1
         5   3.64     osd.5   up  1
         6   3.64     osd.6   up  1
         7   3.64     osd.7   up  1
         8   3.64     osd.8   up  1
         9   3.64     osd.9   up  1
        10   3.64     osd.10  up  1
        11   3.64     osd.11  up  1
    -3  43.68   host ceph-server-2
        12   3.64     osd.12  up  1
        13   3.64     osd.13  up  1
        14   3.64     osd.14  up  1
        15   3.64     osd.15  up  1
        16   3.64     osd.16  up  1
        17   3.64     osd.17  up  1
        18   3.64     osd.18  up  1
        19   3.64     osd.19  up  1
        20   3.64     osd.20  up  1
        21   3.64     osd.21  up  1
        22   3.64     osd.22  up  1
        23   3.64     osd.23  up  1
    -4  43.68   host ceph-server-3
        24   3.64     osd.24  up  1
        25   3.64     osd.25  up  1
        26   3.64     osd.26  up  1
        27   3.64     osd.27  up  1
        28   3.64     osd.28  up  1
        29   3.64     osd.29  up  1
        30   3.64     osd.30  up  1
        31   3.64     osd.31  up  1
        32   3.64     osd.32  up  1
        33   3.64     osd.33  up  1
        34   3.64     osd.34  up  1
        35   3.64     osd.35  up  1
```

De seguida, é necessário configurar as regras de distribuição e replicação de objetos, para isso é necessário modificar o mapa do CRUSH, isto faz-se requisitando o mesmo ao monitor. O mapa CRUSH permite desenhar a hierárquia CRUSH, isto determina como os objetos são distribuídos e replicados pela infraestrutura de armazenamento. Permite

também definir *buckets* (listagem 4.4), ou seja, os nós intermédios da hierárquia CRUSH (listagem 4.5), podemos definir centros de dados, salas, filas, chassis, gavetas, entre outros, como *buckets*; a configuração trata os servidores como *buckets* do tipo *host*. Cada OSD é definido por omissão com um peso equivalente à capacidade da partição de dados que lhe é dedicado, neste caso podemos conferir 3.64 GB de espaço útil.

Listing 4.4: Exemplo definição de *buckets* (ficheiro de configuração *crushmap*).

```
# types
type 0 osd
type 1 host
type 2 chassis
type 3 rack
type 4 row
type 5 pdu
type 6 pod
type 7 room
type 8 datacenter
type 9 region
type 10 root
```

Definiu-se a hierárquia CRUSH (listagem 4.5) em função dos três servidores e respetivo armazenamento, cada sistema anfitrião atua como um *bucket*, um nó intermédio, os OSDs são definidos como *items* do sistema anfitrião. O tipo de *buckets* utilizado é o uniforme, dado que o conjunto de discos que temos à disposição possuem todos os mesmos requisitos (ver secção 3.3.6 para uma descrição detalhada sobre os vários algoritmo). O *bucket* `storage` representa a raiz da árvore CRUSH.

Listing 4.5: Exemplo desenho hierárquia CRUSH (ficheiro de configuração *crushmap*).

```
# buckets
host ceph-server-1 {
  id -6
  # weight 43.680
  alg uniform
  hash 0  # rjenkins1
  item osd.0 weight 3.640 pos 0
  item osd.1 weight 3.640 pos 1
  item osd.2 weight 3.640 pos 2
  item osd.3 weight 3.640 pos 3
  item osd.4 weight 3.640 pos 4
  item osd.5 weight 3.640 pos 5
  item osd.6 weight 3.640 pos 6
  item osd.7 weight 3.640 pos 7
  item osd.8 weight 3.640 pos 8
  item osd.9 weight 3.640 pos 9
  item osd.10 weight 3.640 pos 10
  item osd.11 weight 3.640 pos 11
}
host ceph-server-2 {
```

```
21    id -2
22    # weight 43.680
23    alg uniform
24    hash 0  # rjenkins1
25    item osd.12 weight 3.640 pos 0
26    item osd.13 weight 3.640 pos 1
27    item osd.14 weight 3.640 pos 2
28    item osd.15 weight 3.640 pos 3
29    item osd.16 weight 3.640 pos 4
30    item osd.17 weight 3.640 pos 5
31    item osd.18 weight 3.640 pos 6
32    item osd.19 weight 3.640 pos 7
33    item osd.20 weight 3.640 pos 8
34    item osd.21 weight 3.640 pos 9
35    item osd.22 weight 3.640 pos 10
36    item osd.23 weight 3.640 pos 11
37  }
38  host ceph-server-3 {
39    id -4
40    # weight 43.680
41    alg uniform
42    hash 0  # rjenkins1
43    item osd.24 weight 3.640 pos 0
44    item osd.25 weight 3.640 pos 1
45    item osd.26 weight 3.640 pos 2
46    item osd.27 weight 3.640 pos 3
47    item osd.28 weight 3.640 pos 4
48    item osd.29 weight 3.640 pos 5
49    item osd.30 weight 3.640 pos 6
50    item osd.31 weight 3.640 pos 7
51    item osd.32 weight 3.640 pos 8
52    item osd.33 weight 3.640 pos 9
53    item osd.34 weight 3.640 pos 10
54    item osd.35 weight 3.640 pos 11
55  }
56  root storage {
57    id -1
58    # weight 131.040
59    alg uniform
60    hash 0  # rjenkins1
61    item ceph-server-1 weight 43.680 pos 0
62    item ceph-server-2 weight 43.680 pos 1
63    item ceph-server-3 weight 43.680 pos 2
64  }
```

Após o desenho da hierárquia CRUSH, é necessário escrever as regras de distribuição e replicação, ou regras CRUSH (listagem 4.6). Primeiro, é necessário definir o nome da regra (por exemplo `replicated_ruleset`); definimos um identificador que irá associar a regra a um conjunto de regras (no exemplo, `ruleset 0`); de seguida podemos definir o

mecanismo que irá permitir salvaguardar os dados em caso de falta (`type replicated` ou `type erasure`); podemos definir o número mínimo e máximo de objetos e réplicas; e finalmente, o procedimento para encontrar um nó folha dentro da árvore/hierárquia CRUSH. Estes parâmetros serão ignorados se injetarmos (o Ceph permite mudar dinamicamente os parâmetros) diretamente novos valores nos repositórios.

Listing 4.6: Regras CRUSH (ficheiro de configuração do mapa CRUSH).

```
# rules
rule replicated_ruleset {
  ruleset 0
  type replicated
  min_size 1
  max_size 10
  step take storage
  step chooseleaf firstn 0 type host
  step emit
}
rule erasure-code {
  ruleset 3
  type erasure
  min_size 3
  max_size 20
  step set_chooseleaf_tries 5
  step take storage
  step chooseleaf indep 0 type host
  step emit
}

```

Após definirmos as regras CRUSH, podemos criar repositórios para armazenar os objetos. Para os testes, foram criadas dois repositórios: *replicated* e *erasure coded*; ambos os repositórios possuem mecanismos que permitem tolerância a faltas. O repositório de replicação cria réplicas exatas do objeto, isto é definido através do parâmetro `size`, em que definimos o número de instâncias de um objeto no repositório, para os testes, este parâmetro é alterado ao longo da avaliação, tendo como valores 0, 1 e 2. A *erasure coded* cria dois objetos parciais de 2 MB a partir do objeto original de 4 MB, armazenando segundo as regras CRUSH em nós diferentes, após o qual calcula um objeto de paridade que irá ser armazenado no nó ainda não escolhido.

Por último, fazendo um pedido ao monitor, de uma consulta do estado do *cluster*, o mesmo deve retornar a resposta `HEALTH_OK`, isto significa que todos os intervenientes estão ativos e sem erros, e os repositórios, grupos de distribuição e objetos estão consistentes.

4.5 Resultados e Análise

A fim de entender como o Ceph atua como um sistema de armazenamento distribuído, sob a infraestrutura anteriormente descrita, em primeiro lugar, os testes foram executados com uma configuração mínima, a fim de compreender as capacidades básicas dos OSDs; em segundo lugar, vários discos foram adicionados ao mesmo servidor físico, de forma a determinar a escalabilidade vertical do Ceph; depois, foram instalados mais vinte e quatro OSDs em mais dois servidores físicos (doze OSDs por servidor), o que permite estudar a escalabilidade horizontal do Ceph.

4.5.1 Desempenho de 1 OSD

A configuração mínima do Ceph, é composta por um monitor e um OSD, nesta configuração não existe replicação de dados. De seguida, o *journal* do OSD é movido a partir do mesmo disco, onde os dados estão a ser armazenados, para um disco dedicado, com isso em mente é esperado ver uma redução no tempo de latência. Os últimos testes desta secção introduzem tolerância a faltas, para isso é adicionado um segundo OSD e são criadas regras CRUSH para criar uma réplica por objeto.

Para o objectivo dos testes instalou-se o sistema de ficheiros recomendado para o Ceph, XFS, com a seguinte configuração:

```
osd mkfs options xfs = -f -i 2048
xfs mount options -o rw, inode64, nobarrier, noatime, logbufs=8
```

A instalação do OSD teve em conta a configuração mais simplista, o *journal* e o sistema de ficheiros local no mesmo disco, em partições separadas:

```
$ ceph-deploy -overwrite-conf osd prepare se20:/dev/sdb
```

Os resultados gerados pela ferramenta interna RADOS Bench, permitem revelar a taxa de transferência e latência do OSD, o que por sua vez permite entender os padrões de operações E/S, tanto para um *journal*, como para o sistema de ficheiros local. Os testes de escrita são executados com o comando:

```
$ rados bench -p testpool 1800 write -t 16 -no-cleanup
```

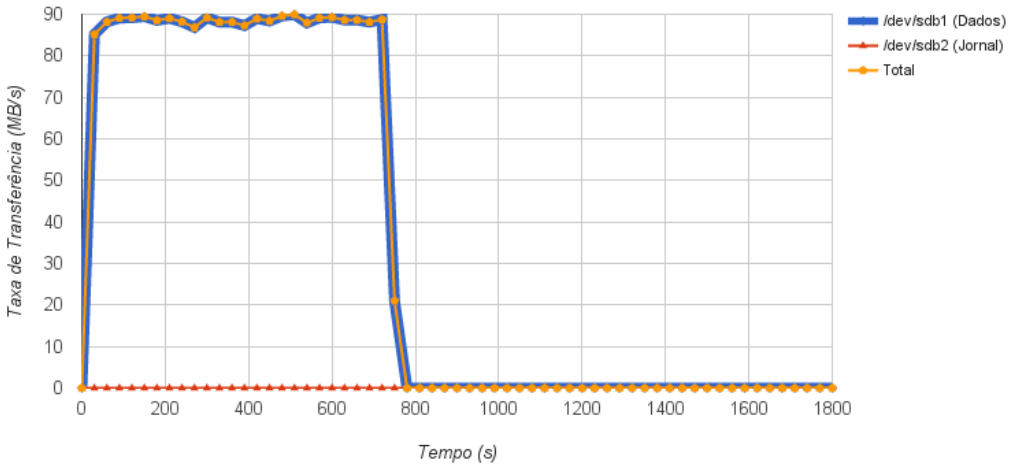
Uma ferramenta de monitorização, Dstat, foi utilizada para observar a transferência de dados em operações de escrita e leitura de duas partições, (`/dev/sdb1` e `/dev/sdb2`). Na Figura 4.10, um OSD escreveu inicialmente para o *journal*, podemos observar a transferência de dados sempre crescente na partição onde o *journal* está armazenado, até que

este foi preenchido ou até este entrar numa fase de sincronização com o sistema de ficheiros, após o qual, a transferência de dados do *journal* começa a ter flutuações muito frequentes, tal como a transferência de dados para o sistema de ficheiros. A largura de banda foi dividida para ambos o *journal* e o sistema de ficheiros.

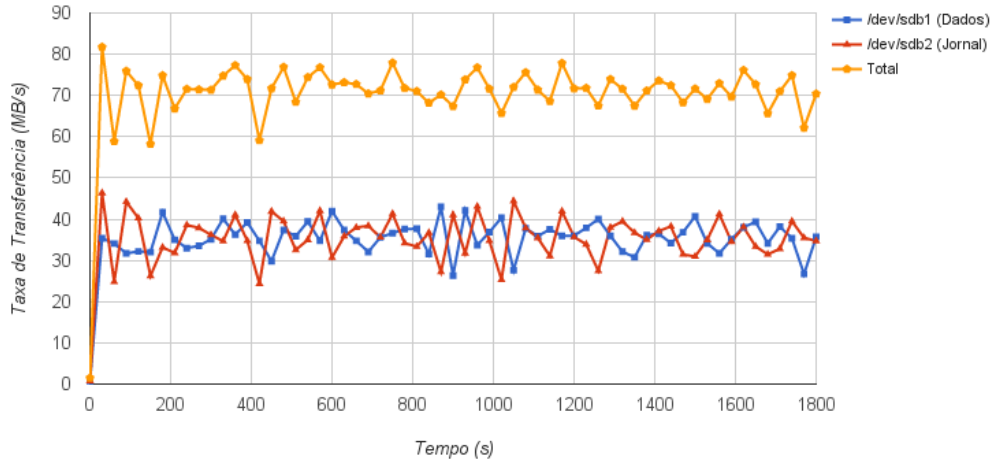
A escrita inicial que é feita apenas no *journal*, passa a ter uma natureza aleatória quando começa a haver sincronização com o sistema de ficheiros, este teste registou uma latência média de 1,88 segundos e uma taxa média de transferência (para o *journal*) de 34 MB/s, em operações de escrita. Leituras sequenciais tiveram melhor desempenho, registando uma taxa de transferência média de 84 MB/s, 750 milissegundos de latência, no entanto, como é esperado não está a aproveitar a largura de banda total. No gráfico de leitura sequencial, é observável que há transferências de dados com a partição que contém o sistema de ficheiros, ao contrário das operações de escrita que escreve primeiro no *journal*, após o qual faz a sincronização com o sistema de ficheiros; leituras aleatórias reforçam esse ponto (há novamente interação apenas com o sistema de ficheiros do OSD), e registou uma taxa de transferência média de 76 MB/s e 837 milissegundos de latência. É importante perceber, que, a fim de executar os testes de leitura aleatórios, um conjunto maior de objetos foi criado, uma vez que com leituras aleatórias o mesmo objeto seria lido várias vezes, atingindo, assim, objetos em *cache*.

A fim de compreender os resultados dos testes de desempenho do Ceph, primeiro devemos entender o seu padrão de E/S, e entender porque utiliza um *journal* enquanto utiliza também um sistema de ficheiros local. Um OSD utiliza um *journal* por duas razões principais: velocidade e confiabilidade. Todas as operações (blocos grandes ou pequenos) podem ser aleatórias quando chegam ao OSD, mas são escritos sequencialmente no *journal*, isso vai melhorar o desempenho, embora quando o sistema de ficheiros sincroniza com o *journal* o desempenho degrada-se para todas as operações de escrita, uma vez que todas as escritas posteriores estão bloqueadas. Um *journal* de um OSD guarda não só metadados, mas também os próprios dados (ao contrário do *journal* do XFS que armazena somente os metadados), de forma síncrona, e só são removidos do *journal* quando a sincronização com o sistema de ficheiros local acaba, ou seja, espera que o comando `syncfs` acabe.

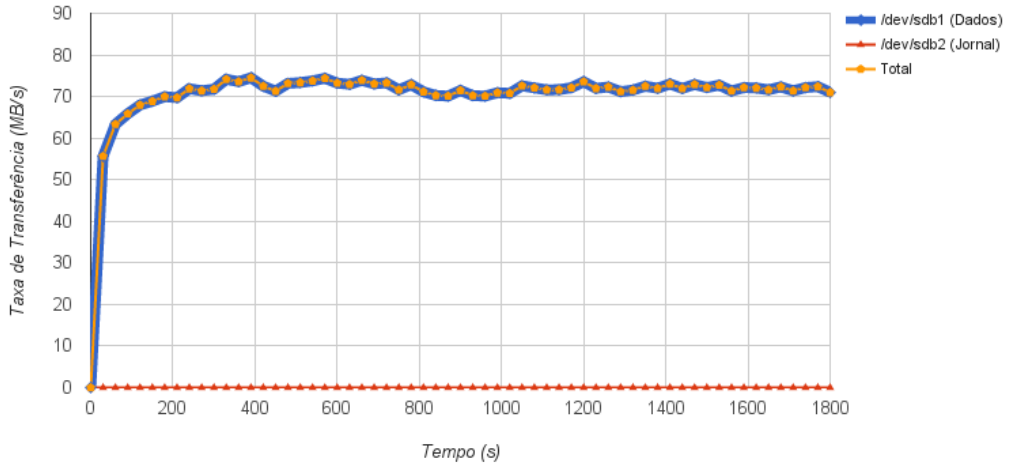
O Ceph tem um padrão concreto de E/S onde todas as modificações são primeiro escritas no jornal de uma forma síncrona e mais tarde submetidas de uma forma assíncrona para o sistema de ficheiros local, se observarmos durante um longo período, podemos concluir que esse padrão é similar a um padrão aleatório em que estamos constantemente a ler e escrever de faixas diferentes no disco; esse padrão aumenta a latência devido à quantidade de tempo a cabeça de uma unidade de disco gasta para se deslocar para um local específico; este tempo inclui: o tempo que gasta para alternar entre as faixas, e o tempo que gasta a localizar o setor desejado. Em conclusão, o disco gasta uma enorme quantidade de tempo em pesquisas de blocos, assim, e tendo seguido a instalação padrão para o OSD, ambos o *journal* e os dados estão no mesmo disco (em partições diferentes), é razoável mover o *journal* para um disco separado e tentar diminuir o tempo



(a) Leitura sequencial.



(b) Escrita sequencial.



(c) Leitura aleatória.

Figura 4.10: Gráficos comparativos de leitura e escrita de objetos num único OSD.

gasto que o disco demora em pesquisas. Assim, podemos concluir que o desempenho é devido ao padrão de escrita e leitura do Ceph.

4.5.2 Journal em Disco Dedicado

A fim de reduzir a latência das operações de escrita e leitura no OSD, uma configuração muito utilizada pela comunidade de administradores de armazenamento Ceph, é mover o *journal* do disco do OSD para um disco dedicado; muitas configurações incluem um disco dedicado para armazenar *journals* de múltiplos OSDs; utilizando uma configuração semelhante, em que movemos o *journal* para um disco SAS idêntico ao utilizado para o sistema de ficheiros do OSD, apesar de não ser um disco mais rápido, há sempre ganhos dado que separamos o tráfego de E/S do *journal*, do tráfego de E/S do sistema de ficheiros XFS (ver a Figura 4.11).

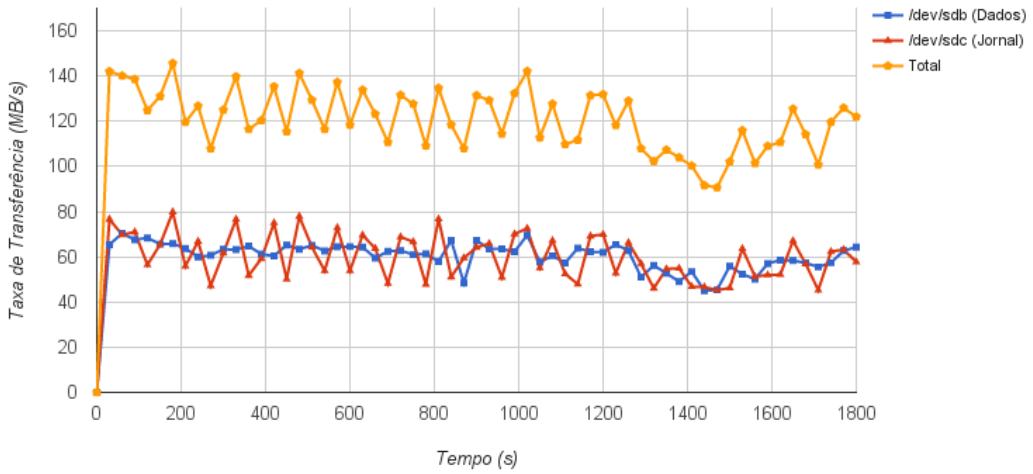


Figura 4.11: Transferência de dados para escritas num OSD, com o *journal* num disco SAS.

Para configurar o OSD a usufruir de um *journal* num disco dedicado é necessário descrever o caminho até ao *journal*, a seguir à localização do sistema de ficheiros, com o comando osd:

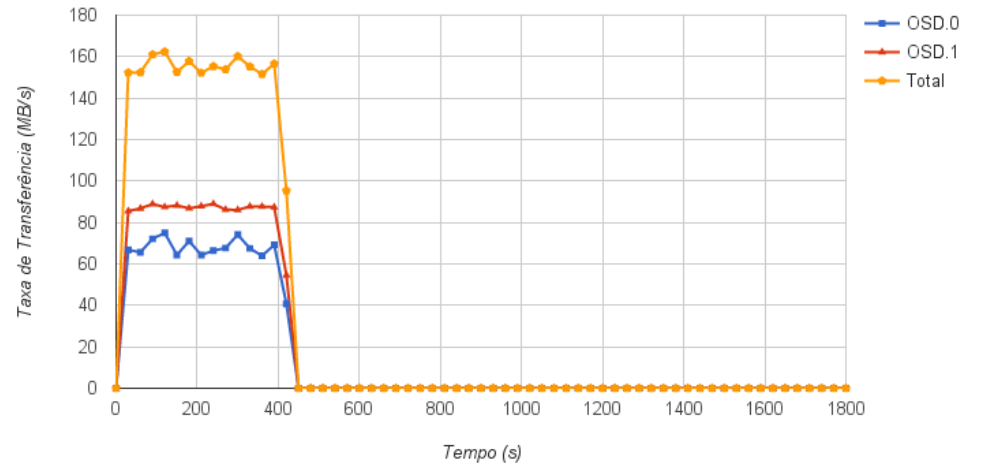
```
ceph-deploy -overwrite-conf osd prepare se20:/dev/sdb:/dev/sdc
```

Admitindo que o disco é dedicado inteiramente ao *journal*, é possível reduzir a latência e aumentar a transferência de dados, 1.11 segundos e 57 MB/s, respetivamente; o ponto negativo, é o facto de ser necessário consumir um disco para o armazenamento do

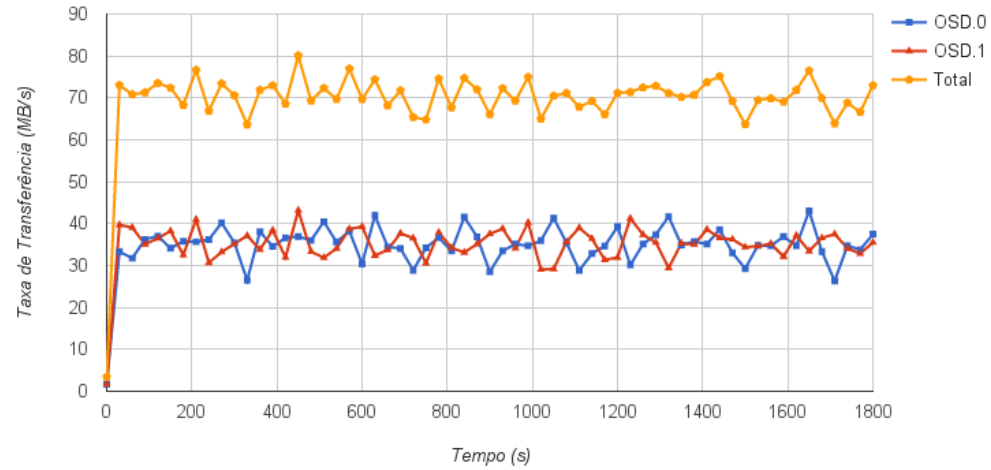
journal para conseguir ter estes resultados, seja com os recursos físicos presentes no ambiente de testes, ou com qualquer outros recursos. Num cenário ideal, discos SATA/SAS de grandes capacidades são utilizados para o sistema de ficheiros dos OSDs e um SSD armazena os *journals* dos respetivos OSDS, mas mesmo neste cenário há limites para a quantidade de *journals* que devemos armazenar no mesmo OSD de modo a manter um bom desempenho.

4.5.3 Replicação de Objetos

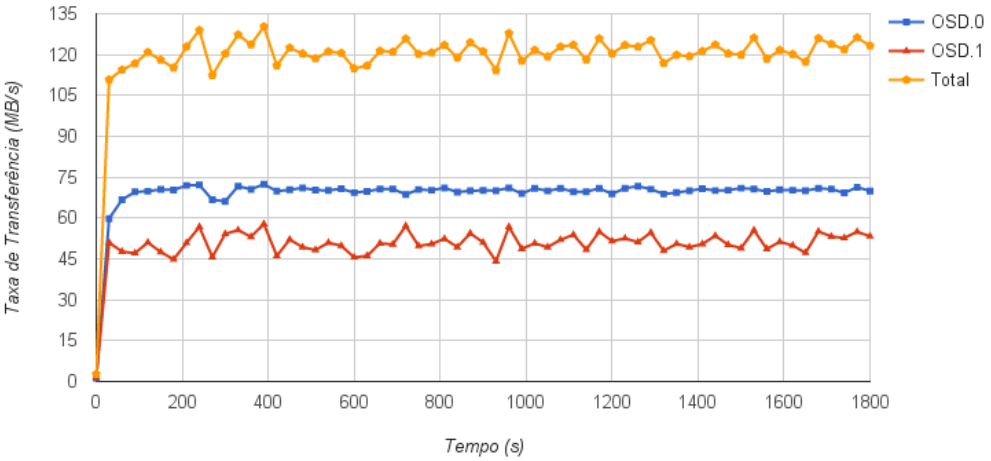
Após estabelecer o desempenho de um OSD, foi necessário introduzir replicação e analisar o impacto que esta tem no desempenho geral (ver a Figura 4.12); para isso, foi necessário configurar dois OSDs, cada um fazendo uma espécie de espelhamento dos objetos do outro disco, equivalente a um volume RAID-1 de dois discos, no entanto, no caso do Ceph, existem objetos originais e respetivas cópias em ambos os discos.



(a) Leitura sequencial.



(b) Escrita sequencial.



(c) Leitura aleatória.

Figura 4.12: Gráficos comparativos de leitura e escrita de objetos num único OSD.

A configuração para estes testes inclui replicação, uma cópia exata de cada objeto, assim, cada OSD acaba por gerar tráfego de replicação; um OSD replica os seus objetos para o outro OSD e vice-versa, os resultados mostram o impacto que a replicação tem.

O teste de escrita sequencial indica que as cargas de trabalho distribuem-se igualmente entre OSDs. Os testes indicam que a replicação de dados não tem impacto significativo, comparando com o teste inicial em que temos um único OSD.

Tendo em conta a distribuição uniforme de cargas e o facto de que a replicação não tem impacto significativo no desempenho, implica que o armazenamento Ceph tem melhor desempenho à medida que adicionamos mais OSDs ao grupo; isto leva também a outra conclusão, para armazenamento Ceph, os discos mais apropriados não tem de ter exatamente grande capacidade, a prioridade é sim aumentar o fluxo de E/S paralelo, para isso deve-se investir em grandes quantidades de discos e definir cada disco como um OSD.

4.5.4 Escalabilidade Vertical

No inicio, ter um sistema que escale, é a melhor solução, aproveitando recursos antigos e adicionando novos, isto irá resultar num melhor desempenho devido ao facto dos discos antigos serem provavelmente o factor limitante; no entanto, à medida que a carga de trabalho aumenta e mais discos são adicionados, os controladores em si tornam-se um gargalo, pois consome mais CPU e RAM. Uma vez que os controladores estão saturados, a única solução é atualizar para outros controladores com especificações mais elevados, uma vez que adicionar mais armazenamento ou através da melhoria da ligação entre o armazenamento e os controladores ou ligação entre o servidor de armazenamento e os clientes não diminuirá a utilização da CPU dos controladores.

Escalabilidade é uma das características mais relevantes de que um sistema de armazenamento distribuído como Ceph é obrigado a ter. Um método comum de atualização de um servidor de armazenamento, é atualizar os seus recursos, quer substituindo os seus recursos por melhores, ou pelo aumento do número de recursos; este tipo de expansão é conhecido como escalabilidade vertical e horizontal. Apenas soluções de escalabilidade vertical não são indicadas para ambientes de computação em nuvem, uma vez que os requisitos mudam à medida que novas aplicações são executadas; Ceph tira proveito de uma arquitetura verticalmente escalável apenas como uma parte de sua solução.

Os primeiros testes de escalabilidade centraram-se na adição de mais recursos a um nó do sistema, neste caso, mais discos a um servidor. Limitámos a escalabilidade a uma caixa de discos, por isso adicionámos até 12 discos. A configuração do *cluster* exclui a replicação de objetos. Os testes, realizados com o RADOS Bench, escrevem e lêem blocos de 4 MB, tendo o tamanho do *job* mais de 128 GB, o dobro da memória do *cluster*; os mesmos testes foram executados com o nº de *threads* a variar entre 1 e 32 *threads* (mais especificamente 1, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24 e 32), e também testámos nº de *threads* igual ao dobro do nº de discos, isto porque cada OSD executa duas *threads*.

Como pode ser observado pelas Figuras 4.13, não há uma grande disparidade quando o número de *threads* a gerarem E/S não está alinhado com o número de *threads* OSD, isto é quando temos mais *threads* a fazerem pedidos do que *threads* a servirem, conseguimos ganhos provenientes de uma melhor utilização do tempo de CPU e utilização dos discos.

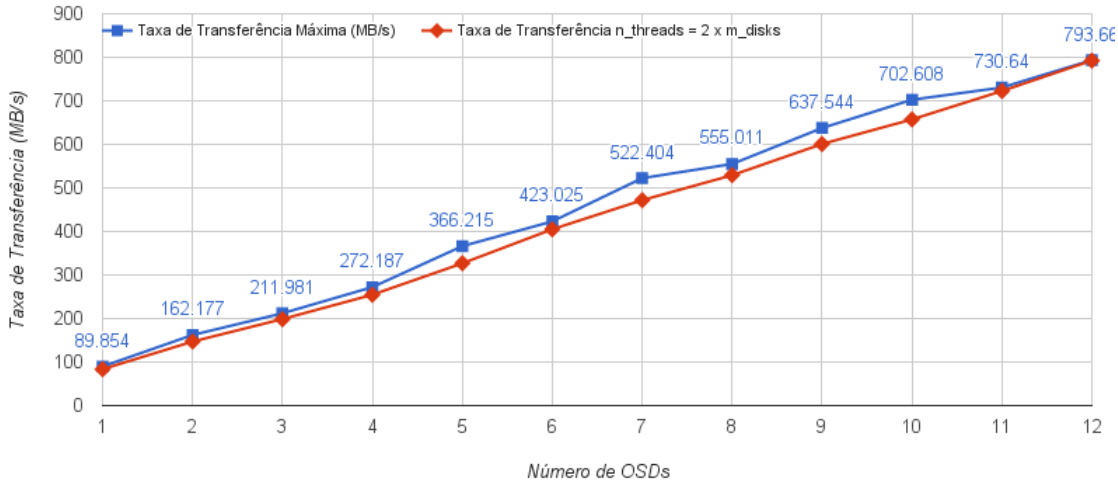


Figura 4.13: Gráfico comparativo para leituras sequenciais, apresentando a taxa de transferência máxima e taxa de transferência conseguida igualando o número de *threads* de E/S com o número de *threads* OSDs.

A adição do 2º OSD leva a um incremento de cerca de 72 MB/s para operações de leitura (90 MB/s passou para 162 MB/s). A taxa de transferência continua a crescer, situando-se no seu valor mais alto com 12 OSDs, com 794 MB/s para leituras. Como podemos ver a diferença entre as duas retas da Figura 4.14 é mínima, isto deve-se ao fato de que por OSD são executadas duas *threads* para servirem pedidos de E/S. Portanto, esta é uma boa visão da largura de banda disponível, a reta com os melhores resultados apresenta melhor desempenho devido ao tempo de latência ser alto, utilizando maior concorrência permite-nos tomar melhor partido dos 16 *cores* por servidor físico.

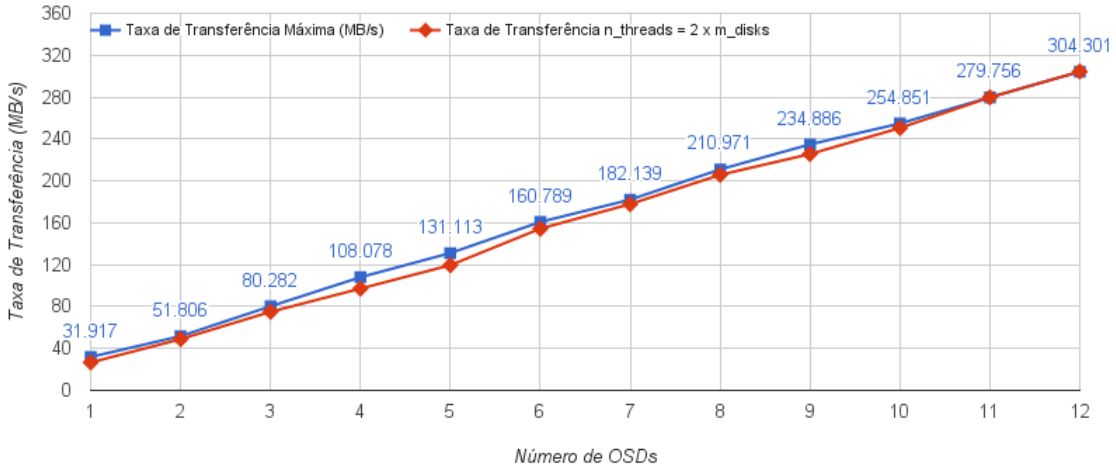


Figura 4.14: Gráfico comparativo para escritas sequenciais, apresentando a taxa de transferência máxima e taxa de transferência conseguida igualando o número de *threads* de E/S com o número de *threads* OSDs.

Para operações de escrita (Figura 4.15), a adição do 2º OSD leva a um incremento de 20 MB/s (passando de 32 MB/s para 52 MB/s); para o 3º OSD aumenta de 52 MB/s para 80 MB/s, um incremento de 28 MB/s; este crescimento da largura de banda continua até aos 12 discos com incrementos entre 20 MB/s e 30 MB/s, sendo a melhor taxa de transferência conseguida com 12 OSDs. Os testes em que o nº de *threads* do lado de cliente são equiparadas ao dobro das *threads* do *cluster* (ou seja para cada *thread* a gerar operações de E/S existe uma *thread* de um OSD), podemos constatar que consegue um valor muito aproximado comparando com os melhores resultados conseguidos; pelo que podemos concluir que para escritas, o desempenho do *cluster* é garantidamente crescente enquanto o nº de clientes for inferior ao dobro do nº total de *threads* OSD (relembramos que 1 OSD executa 2 *threads*); é possível configurar o nº de *threads* que cada OSD executa, pelo que é possível manter o mesmo desempenho mesmo se o nº de pedidos de clientes aumentar.

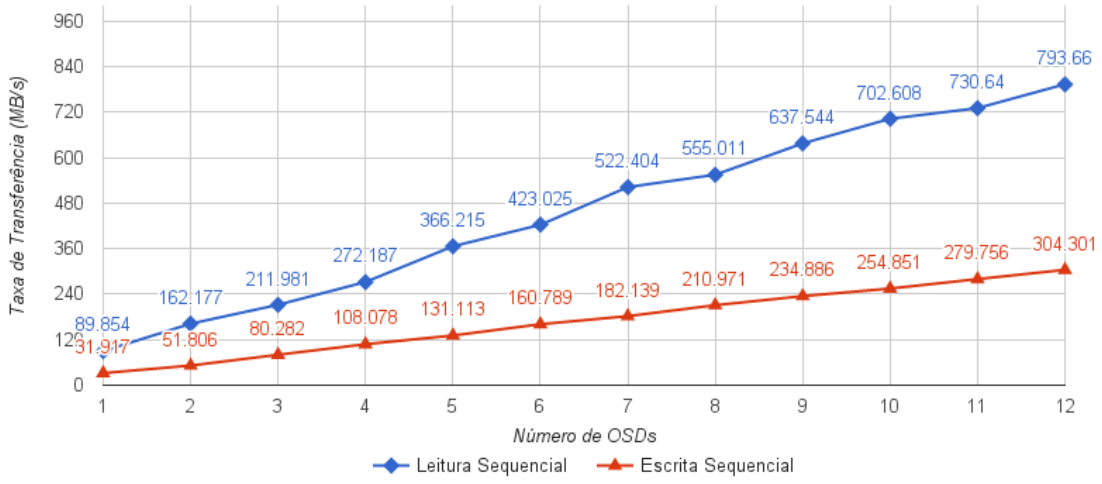


Figura 4.15: Gráfico comparativo para escritas e leituras sequenciais, apresentando a taxa de transferência máxima e taxa de transferência conseguida igualando o número de *threads* de E/S com o número de *threads* OSDs.

A taxa de transferência para operações de leitura aumentou linearmente de 90 MB/s até 794 MB/s, representa um crescimento gradual da largura de banda do Ceph; do mesmo modo, operações de escrita também apresentam um crescimento gradual, linear, apesar de que (como pode ser visto na Figura 4.15) tem valores muito mais baixos comparando com as respetivas operações de leitura, de 32 MB/s aumentou até 304 MB/s.

Os resultados apresentam valores esperados, cada OSD está a utilizar cerca de 34 % (para escritas) e 42 % (para leituras) da largura de banda máxima conseguida na caracterização de 12 discos em RAID-0, embora, como foi dito anteriormente, isto deve-se ao *overhead* introduzido pelos padrões de escrita e leitura dos objetos, e ainda, ao facto de termos um sistema de ficheiros local (XFS).

A adição de novos discos leva o armazenamento a escalar linearmente e permite-nos eventualmente chegar ao desempenho desejado; este facto, deve-se à sua arquitetura ser composta por unidades autossuficientes. Cada OSD, trabalha de uma forma independente, contactando os monitores quando apenas têm informação desatualizada do mapa do *cluster*; para além disso o sistema delega a responsabilidade de migração de dados, replicação, deteção e recuperação de faltas de OSDs. Com o tempo as necessidades mudam, e por isso é crucial que a aplicação adapte-se facilmente, neste caso escalando verticalmente, seja incrementando/decrementando o número de *threads* OSD, seja adicionando ou removendo discos do servidor.

4.5.5 Escalabilidade Horizontal

A seu dado tempo, a capacidade e o desempenho das infraestruturas em ambientes de produção deixam de servir os requisitos das várias aplicações, uma vez que existe um limite para o quanto se pode atualizar o mesmo servidor, adicionando novos recursos, com isto, estamos a falar de escalabilidade horizontal.

Escalar horizontalmente não é necessariamente um novo conceito, no entanto, é a estratégia ideal para suportar a evolução das infraestruturas de armazenamento de grandes volumes de dados.

O sistema Ceph segue uma arquitetura que escala horizontalmente, em que cada servidor é carregado com discos, e que possibilita a adição de mais servidores de forma a melhorar a capacidade de armazenamento e desempenho do sistema de armazenamento. Uma das maiores vantagens de utilizar o Ceph é o facto do armazenamento escalar de acordo com os requisitos; a adição de novos servidores não tem qualquer impacto no sistema, exceto a migração de dados que é observada no momento em que o novo OSD toma a responsabilidade de armazenar todos os objetos que são mapeados com o seu identificador.

Os testes de escalabilidade horizontal centraram-se na instalação do Ceph, num segundo nó (totalizando o número de discos em 24) e num terceiro nó (36 discos).

Os testes iniciais de escalabilidade horizontal (Figura 4.16), em que não há replicação de objetos, mostram que o *cluster* escala horizontalmente, ou seja, com a adição de mais máquinas físicas podemos ver o desempenho a escalar. Para escritas, observam-se resultados semelhantes ao esperado, sendo 304 MB/s, 612 MB/s e 947 MB/s para 1, 2 e 3 nós; para leituras, obteve-se 794 MB/s, 1442 MB/s e 2012 MB/s, sendo estes resultados abaixo do resultado, o que significa uma perda de desempenho de 9 % para 2 nós e 15 % para 3 nós.

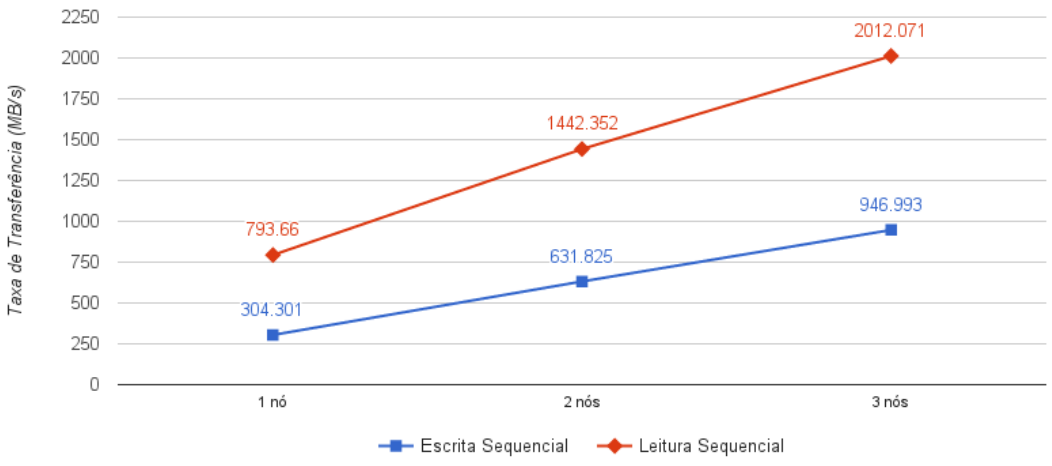


Figura 4.16: Gráfico comparativo de repositórios replicados para 1,2 e 3 nós.

Nas situações em que introduzimos replicação de objetos (Figura 4.17), tendo uma réplica por objeto, podemos ver que para as escritas, obteve-se 335 MB/s e 773 MB/s para 2 e 3 nós, para leituras obteve-se 1401 MB/s e 2009 MB/s para 2 e 3 nós.

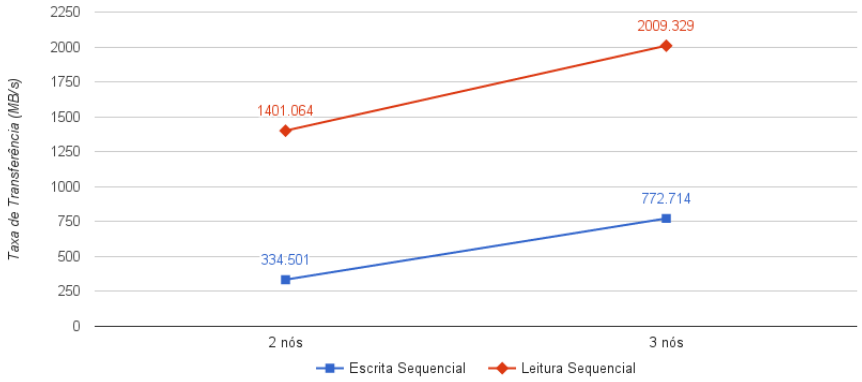


Figura 4.17: Gráfico comparativo de repositórios com replicação, 1 réplica por objeto, para 2 e 3 nós.

A replicação, seguramente não tem nenhum impacto em operações de leitura; o caso muda significamente para escritas (Figura 4.18), em que se regista 947 MB/s quando não incluímos replicação, 773 MB/s quando replicamos 1 vez o objeto e 336 MB/s quando replicamos 2 vezes cada objeto; sendo visível um maior agravamento do desempenho no último caso, verificando-se que o desempenho decrementa por um factor de 3. O impacto no desempenho, deve-se ao facto de ser necessário ao OSD primário esperar pela criação dos objetos no OSD secundário e terciário, para poder concluir a operação com sucesso.

Por último, é reunido os resultados de testes anteriores e são comparados (Figura 4.19), mas também é introduzido um novo tipo de mecanismo para tolerância a faltas, através do mecanismo *erasure coding*. Uma *pool* com *erasure coding* em que dividimos um objeto em objetos de dados ou *data chunks*, sendo calculado m objetos de paridade (ou *coding chunk*), para permitir a tolerância a falta de um disco, juntamente com k-m objetos onde irão ser guardados os dados efetivos (k é por isso mesmo o total de discos a serem utilizados). A utilização de repositórios com *erasure coding*, implica que é necessário menos espaço de armazenamento, comparando com replicação de objetos , neste caso em especifico precisa de 33

Como podemos ver, o desempenho de leituras para 2 nós com replicação e para 3 nós com *erasure coding* é semelhante, 335 MB/s e 336 MB/s, , apesar de estarmos com mais uma máquina, as leituras incidem sobre os nós responsáveis por guardar os objetos de

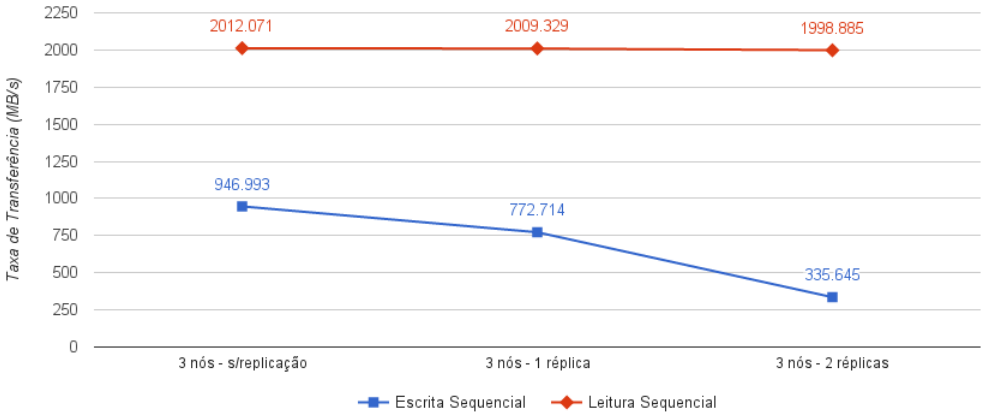


Figura 4.18: Gráfico comparativo de repositórios com replicação, 1 réplica por objeto, para 2 e 3 nós.

dados; estes dois casos também são semelhantes para escritas, devendo-se novamente ao facto de mesmo com 3 nós, o facto de ser necessário o cálculo do objeto de paridade leva a um impacto muito grande no desempenho. Os casos, em que temos um *cluster* formado por 3 nós e um repositório replicado com 1 ou 2 réplicas por objeto, são semelhantes apenas em leituras, 2009 MB/s e 1998 MB/s, pois neste caso a operação é exatamente a mesma; para escritas o desempenho diminui de 773 MB/s e para 336 MB/s, quando aumentamos o nível de replicação, devido ao facto de ser necessário contactar um 3º servidor e enviar uma operação de escrita com esse objeto.

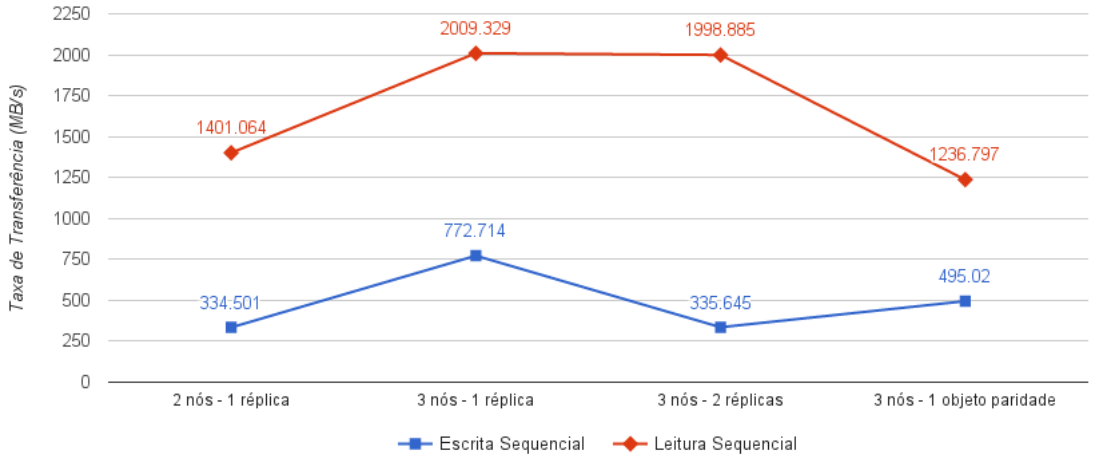


Figura 4.19: Gráfico comparativo de repositórios com replicação e com *erasure coding*.

Em conclusão, podemos observar na tabela 4.5 que as diversas configurações acarretam com vantagens e desvantagens, a nível de tolerância de faltas e de desempenho,

Config.	Nós	*Pool*	Tol. a Falhas (discos)	Escritas (MB/s)	Leituras (MB/s)
1	1	Replicada	0	304.301	793.66
2	2	Replicada	0	650.076	1443.067
3		Replicada	1	334.501	1401.064
4	3	Replicada	0	946.993	2012.071
5		Replicada	1	772.714	2009.329
6		Replicada	2	335.645	1998.885
7		*Erasure Coded*	1	495.02	1236.797

Tabela 4.5: tabela comparativa do desempenho de vários tipos de *pools*.

cabe ao administrador do armazenamento em questão decidir qual o aspeto a que deve dar mais prioridade; caso decida dar prioridade à utilização de espaço em deterioração de tolerância a faltas, deve proceder com a configuração número 4, obterá o melhor proveito da largura de banda; caso decida dar prioridade à utilização de espaço mantendo a tolerância a falta de 1 disco, configuração número 7, deverá considerar que o sistema irá operar com o mesmo desempenho que um sistema com dois nós (sem replicação), pois para cada operação, um dos três nós estará a calcular e armazenar o objeto de paridade; caso o administrador priorize a segurança de dados (isto é, priorizar a recuperação de dados em caso de falta), situação número 6, deverá ter em conta que esta configuração consome mais espaço útil em discos que qualquer outra configuração anunciada; por fim, se o administrador quer um equilíbrio entre desempenho e utilização de espaço útil dos discos, deve optar pela configuração número 5. Para finalizar, podemos através dos dados apresentados, que o Ceph escala linearmente, dado que o desempenho aumenta em conformidade com o número de nós.

4.5.6 Desempenho de um Subsistema de Virtualização

O último conjunto de testes de desempenho destina-se a avaliar os tempos de inicialização e encerramento de máquinas virtuais (VMs), operações típicas no Nova, serviço de computação do OpenStack. Para facilidade de exposição, reintroduzimos a infraestrutura de testes com a Figura 4.20, destacando agora os clientes Ceph (*blades* IBM HS21) que correm o CentOS 6.5 e o hipervisor KVM.

Como já anteriormente descrevemos os servidores Ceph (os `ceph-server` na figura) e respetivos armários de armazenamento, descreveremos agora apenas os clientes Ceph (`ceph-client` na figura): cada um é uma *blade* IBM BladeCenter HS21 com 2 CPUs Quad-core Intel Xeon L5420 a 2.5 GHz, 24 GB RAM, e disco interno onde está instalada uma distribuição Linux CentOS 6.5 com o hipervisor KVM.

Seguem-se os principais comandos para a gestão ambientes KVM através do virsh.

```
$ virsh -c qemu:///system
```

Para iniciar uma sessão no KVM pelo virsh é necessário executar esta operação

inicial de modo a criar uma conexão com o *daemon* libvirtd.

```
$ virsh create configuration_file.xml
```

A criação de máquinas virtuais é feita a partir de XML, isto possui vários benefícios associados, um dos quais é obviamente a criação de VMs a partir de XML de outras VMs.

```
$ virsh dumpxml [domain-id, domain-name or domain-uuid]
```

Extração da definição de uma VM para um ficheiro XML a partir de uma VM existente.

```
$ virsh define configuration_file.xml
```

Definição de um sistema convidado através de um ficheiro de configuração em XML.

```
$ virsh undefine [domain-name, domain-id]
```

Remoção da definição de um sistema convidado, caso este esteja a executar, irá transformar-se numa instância efémera, ou seja, no momento em que o estado da VM passe a inativo, o ficheiro de configuração é removido.

```
$ virsh setvcpus [domain-name, domain-id or domain-uuid] [count]
```

Através do ficheiro de configuração XML é possível a definir sistemas convidados que utilizem a funcionalidade descrita como Symmetric Multi-Processing (SMP), o que permite utilizar um número arbitrário de CPUs do sistema anfitrião, cada CPU é mapeado para o sistema convidado como virtual CPU (ou vCPU); é possível ainda definir afinidade para com os CPUs do sistema anfitrião.

```
$ virsh setmem [domain-id or domain-name] [size]
```

A definição de *hardware* virtual permite definir a memória (ou vRAM) do sistema convidado, sendo reservada a partir da memória do sistema anfitrião, sendo possível ainda com o KVM (visto que VMs são meros processos) utilizar memória Swap.

```
$ virsh net-define [XML file]
```

A rede é definida em KVM através da criação de virtual network interface cards (vNIC), sendo este dispositivo virtualizado pelo QEMU e gerido para que os pacotes possam ser reencaminhados do NIC físico para o vNIC.

```
$ virsh attach-disk example-vm -source
/var/lib/libvirt/images/vm-image.img
-target vdb -persistent
```

Por fim, é necessário falar na definição de discos virtuais (virtual disks ou vDisks), os quais são necessários sejam para serem utilizados como discos de sistema sejam

para serem utilizados para armazenamento secundário. Com o KVM é possível a definição do *vDisk* à custa de um ficheiro ou mesmo através de um disco físico (LUN); o ficheiro é normalmente conhecido como imagem, existindo em vários formatos, tais como: *RAW*, permite uma melhor utilização de espaço, apenas ocupa a quantidade ocupada em disco virtual; os formatos QEMU qcow e qcow2 e cow, formato VMware vmdk; e, VirtualBox vdi.

```
$ virsh start [domain-id or domain-name]
```

Comando executado para inicializar uma VM previamente inactiva.

```
$ virsh shutdown [domain-id or domain-name]
```

Comando executado para encerramar uma VM previamente activa.

Outros comandos que dada a sua relevância, merecem ser referênciados:

```
$ virt-clone -original [domain-name]
-name [clone-domain-name] -file [disk-path]
```

Para clonagem de VMs é utilizado outra CLI chamada virt-clone.

```
$ virsh migrate [domain-name] qemu+ssh://[Hostname or IP]
```

Executado para migração de VMs, sendo necessário indicar o *fully qualified domain name* do sistema anfitrião para o qual queremos migrar a VM.

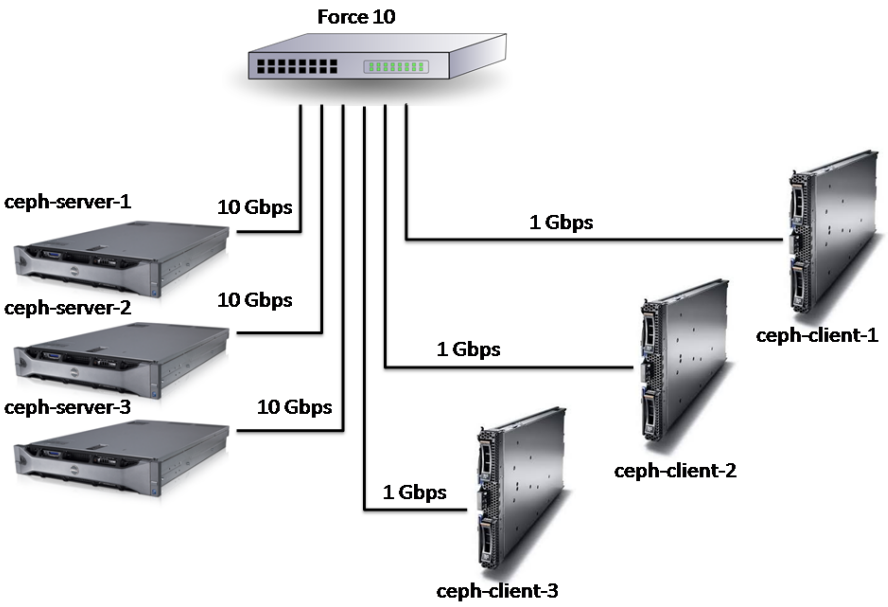


Figura 4.20: Rede da infraestrutura de testes.

O Ceph constitui-se então como um repositório (*pool*) de armazenamento onde estão as VMs que são usadas nos testes, testes estes que consistem muito simplesmente em medir o tempo necessário para arrancar e desligar nas *blades* (clientes Ceph) um grupo de VMs. Os testes começam com 1 VM e vão sendo repetidos até se atingir um total de 32 VMs por *blade*, o que corresponde a um total de 96 VMs quando cada uma das 3 *blades* está a executar 32 VMs. O tempo de inicialização de uma VM é o que decorre entre o instante no qual é desencadeado o arranque (*boot*) e a receção do primeiro pacote ICMP de ping efetuado pela VM; o tempo de encerramento é o que decorre entre o instante no qual é desencadeado o encerramento (*shutdown*) e o tempo registado no *log* do hipervisor.

As VMs foram todas obtidas por clonagem completa (*full clone*), técnica na qual o espaço de armazenamento gasto por cada clone é igual ao do *template* (imagem) que lhe deu origem. Os valores de referência para os recursos associados à imagem base são os seguintes: o espaço ocupado pela imagem no sistema de armazenamento é de 10 GB, exatamente a mesma dimensão do disco virtual de cada VM - ou seja, não foi usado *thin provisioning* (aprovisionamento esparso) para a criação do disco virtual; o número de CPUs da VM é de 1, e a RAM de 256 MB; a VM tem uma única placa de rede, de tipo Gigabit Ethernet.

Como pode ser observado na Figura 4.21, a inicialização de uma máquina virtual demorou 22 segundos, durante os quais passaram 82MB de tráfego na interface da máquina hospedeira (cliente KVM); tal tráfego deve-se maioritariamente ao carregamento do *kernel*, bibliotecas e utilitários, e à execução de serviços que acedem ao disco (virtual) da VM.

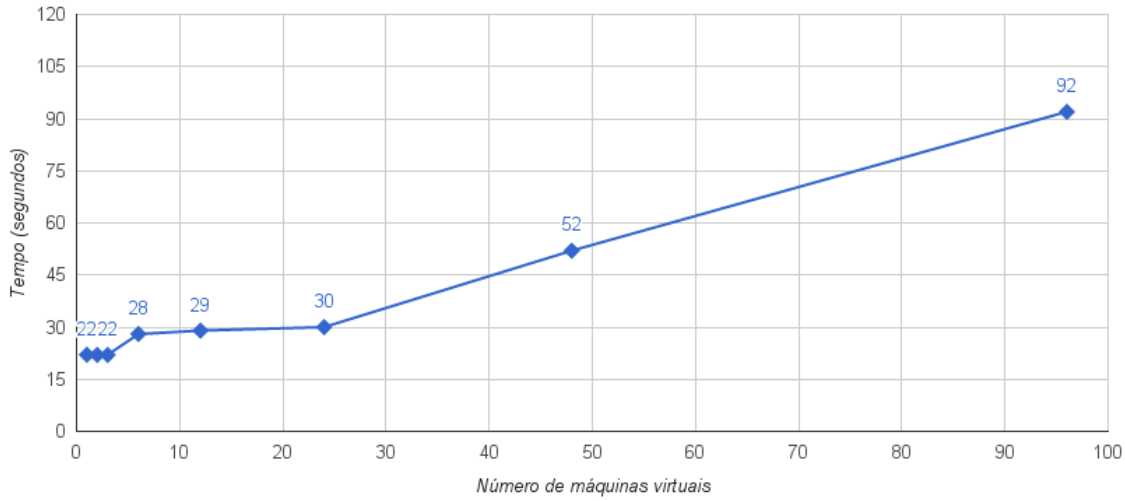


Figura 4.21: Tempos de inicialização de máquinas virtuais.

Conforme se pode observar, o tempo de inicialização manteve-se em 22s até às 3 VMs (i.e., enquanto a execução não excedeu uma VM por hospedeiro); aumentou para 28s

para 6 VMs (com 2 VMs por hospedeiro), e 29 e 30s para a inicialização de 12 e 24 VMs (com 4 e 8 VMs por hospedeiro). Por fim, para 48 e 96 VMs (16 e 32 VMs por hospedeiro, respectivamente) os tempos de inicialização aumentaram para 52 e 92s. O tráfego observado nas interfaces físicas das *blades* nos diversos testes foi sempre absolutamente proporcional ao número de VMs, culminando com o valor aproximado de aproximadamente 2.6 GB por hospedeiro (32 VMs x 82 MB = 2624 MB) no teste de 96 VMs.

A Figura 4.22 apresenta os tempos de encerramento de VMs, para configurações idênticas às usadas nos testes de *boot*; assim, trata-se de realizar o *shutdown* de entre 1 e 96 VMs, distribuídas pelas três *blades*. O tempo de *shutdown* foram: de 8s para 1 VM, gerando um tráfego de cerca de 3.7 MB na *interface* do hospedeiro; 8s, ainda, para 2 e 3 VMs; 10s para 6 e 12 VMs (2 e 4 VMs por *blade*); 12s para 24 VMs (8 VMs por *blade*); e 16s para 48 VMs (16 VMs por *blade*). Finalmente, para 96 VMs (32 VMs por *blade*) nota-se um aumento no declive da linha, no gráfico, e o tempo aumentou para 32s. Confirmou-se que o tráfego gerado na placa de rede de cada *blade* (cliente) mantém-se proporcional ao número de VMs, sendo neste último teste de aproximadamente 108 MB (32 x 3.7 MB = 118.4).

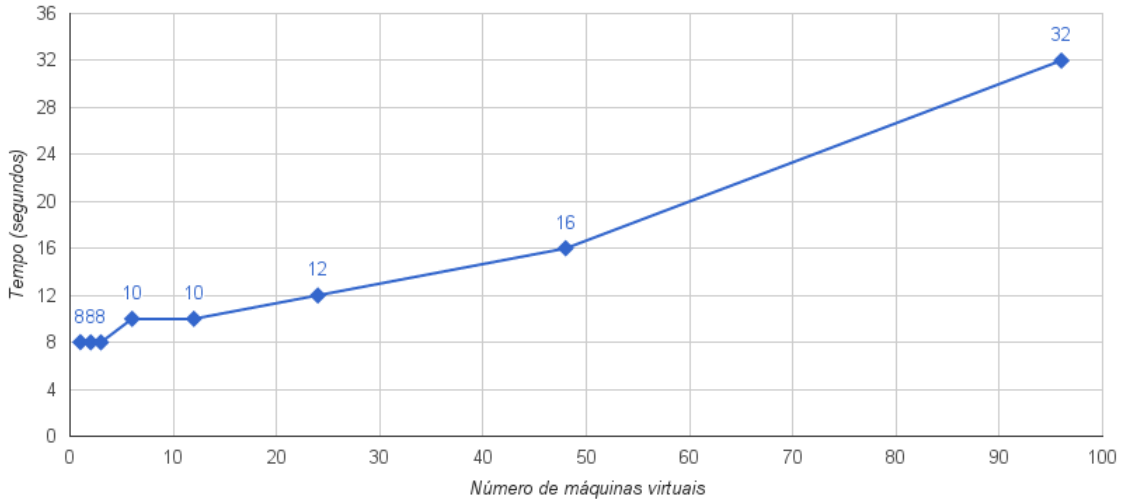


Figura 4.22: Tempos de encerramento de máquinas virtuais.

5

Conclusões e Trabalho Futuro

5.1 Conclusões

O conjunto de VMs cujo número fomos fazendo crescer nos últimos testes realizados na secção anterior pode representar ambientes tão diversos como um "cluster HPC", em que as VMs vão usar processamento paralelo para acelerar a resolução de um problema de grande escala, ou uma arquitetura n-tier, típica de um ambiente empresarial de IT com múltiplos níveis de serviço interdependentes, e.g., servidores Web, aplicacionais e de bases de dados.

Recorrendo aos conhecimentos de que dispomos e a algumas simplificações, porventura excessivas pois não suportadas por análises e/ou experimentações que as consubstanciem, podemos dizer que o tempo de boot de um conjunto de VMs pode ser decomposto em: a) tempo necessário para carregar a imagem da VM em memória do hospedeiro; b) tempo necessário para fazer o boot do núcleo do sistema de operação; e, c) tempo necessário para arrancar os serviços base do SO (que constituem o conjunto mínimo para disponibilizar, por exemplo, um login de utilizador.

Assim, observando a Figura 4.21 podemos, ao ver como evolui o tempo necessário para arrancar um certo número de VMs, destacar alguns aspetos:

- Até 8 VMs por host, o tempo cresce muito lentamente: 22s, 26s, 29s e 30s (para 1, 2, 4 e 8 VMs);
- A partir de 8 VMs por host, o tempo cresce abruptamente: 52s e 92s (para 16 e 32 VMs).

Como sabemos que (a) será o tempo necessário para carregar 82 MB (tráfego medido na secção 4.5.6) e que dispomos nas blades de um NIC de 1 Gbps, podemos admitir

que o carregamento da imagem demorará cerca de 1s; teremos então aproximadamente 21s para (b) e (c). Até 8 VMs/host a expressão produz valores bastante aproximados, o que já não acontece acima de 8 VMs/host. A explicação é simples: acima desse valor, ultrapassa-se o número de cores da blade (recordemos que tem 2 CPUs quad-core), pelo que há um distribuir do tempo de CPU real pelos CPUs virtuais, afectando-se os fatores (b) e (c).

Assim, o micro-benchmark realizado e que avalia o tempo de startup e shutdown de "clusters" de VMs mostra que a infraestrutura de armazenamento e os servidores, associados a um sistema de armazenamento baseado em objetos suportam eficazmente um ambiente de virtualização enquanto os recursos dos servidores de virtualização (aqui representados pelas blades IBM HS21) forem suficientes para as necessidades das VMs - quer em termos de cores, quer em termos de RAM.

Quando os recursos são mais escassos, como no caso dos testes com um número de VMs bastante superior ao dos cores, há um crescimento abrupto dos tempos de inicialização que, numa experiência que realizamos com 120 VMs (correspondendo a 40 VMs/host), redundou em "soft lockups", que são mensagens de aviso emitidas pelo núcleo do Linux indicando que determinadas tarefas que o núcleo realiza periodicamente não foram atempadamente desencadeadas.

Resta-nos terminar referindo apenas que, ainda que o objetivo fundamental deste trabalho fosse colocar o Ceph "ao serviço" de uma infraestrutura de cloud IaaS OpenStack, consideramos que os resultados aqui conseguidos são direta e totalmente aplicáveis a uma tal situação, já que o serviço computacional Nova do OpenStack é essencialmente o ambiente que usamos, Linux + KVM, "apenas" acrescido de middleware de integração nos restantes serviços (autenticação, etc.) do OpenStack.

5.2 Trabalho Futuro

Ainda que os resultados sejam muito encorajadores, estamos longe de ter realizado uma avaliação completa. Para caracterizar melhor os serviços prestados pela infraestrutura e subsistema Ceph precisamos de avaliar situações nas quais as VMs realizam I/O considerável; a caracterização pode ser efetuada recorrendo a benchmarks de SGBDs (TPC-E, TPC-H), Hadoop, benchmarks de sistemas de ficheiros (IOzone, Bonie++) e outros.

Bibliografia

[AK07] D. L. U. L. e. A. L. Avi Kivity Yaniv Kamay. "kvm: the Linux Virtual Machine monitor." Em: *Proceedings of the Linux Symposium*. Ottawa, Canadá, 2007.

[BDFHHHNPW03] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt e A. Warfield. "Xen and the Art of Virtualization". Em: *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*. SOSP '03. Bolton Landing, NY, USA: ACM, 2003, pp. 164–177. ISBN: 1-58113-757-5. DOI: 10.1145/945445.945462. URL: http://doi.acm.org/10.1145/945445.945462.

[BZ02] P. J. Braam e R. Zahir. "Lustre: A scalable, high performance file system". Em: *Cluster File Systems, Inc* (2002).

[Cep] *Ceph*. URL: http://ceph.com/ (acedido em 13/05/2014).

[Cin] *Cinder Basics*. URL: http://cloudarchitectmusings.com/2013/11/18/laying-cinder-block-volumes-in-openstack-part-1-the-basics/ (acedido em 01/12/2015).

[ELM04] R. J. H. e Ethan L. Miller. "Replication Under Scalable Hashing: A Family of Algorithms for Scalable Decentralized Data Distribution". Em: *Proceedings of the 18th International Parallel & Distributed Processing Symposium (IPDPS 2004)*. Abr. de 2004.

[GLM13] B. D. e Cyril Séguin e Gael Le Mahec. "Analysis of Six Distributed File Systems". Em: (2013). hal-00789086, version 1.

[Glua] *Gluster documentation*. URL: http://www.gluster.org/community/documentation/index.php/Main_Page (acedido em 08/04/2014).

[Glub] "Gluster File System Architecture". Em: 2010. URL: ftp://ftp.pku.edu.cn/open/filesystem/GlusterFS/doc/Gluster_Architecture.pdf.

[Jr.95] R. J. J. Jr. *Hash Functions for Hash Table Lookup*. 1995-1997. URL: http://burtleburtle.net/bob/hash/evahash.html (acedido em 07/02/2015).

[KKLLL07] A. Kivity, Y. Kamay, D. Laor, U. Lublin e A. Liguori. "kvm: the Linux Virtual Machine Monitor". Em: *Proceedings of the Linux Symposium*. Vol. 1. Ottawa, Ontario, Canada, jun. de 2007, pp. 225–230. URL: http://linux-security.cn/ebooks/ols2007/OLS2007-Proceedings-V1.pdf.

[Lam98] L. Lamport. "The Part-time Parliament". Em: *ACM Trans. Comput. Syst.* 16.2 (mai. de 1998), pp. 133–169. ISSN: 0734-2071. DOI: 10.1145/279227.279229. URL: http://doi.acm.org/10.1145/279227.279229.

[Lam01] L. Lamport. "Paxos Made Simple". Em: *SIGACT News* 32.4 (dez. de 2001), pp. 51–58. ISSN: 0163-5700. DOI: 10.1145/568425.568433. URL: http://research.microsoft.com/users/lamport/pubs/paxos-simple.pdf.

[Lop09] P. Lopes. "A Shared-Disk Parallel Cluster File System". Tese de doutoramento. Portugal, 2009.

[Oli07] T. d. A. R. e. A. S. C. e. B. d. O. S. Márcio Parise Boufleur e Matheus Anversa Viera e Rodolfo Leffa de Oliveira. "Avaliação do Sistema de Arquivos Distribuído GlusterFS". Em: (2007).

[Opea] *OpenStack*. URL: http://www.openstack.org/ (acedido em 18/06/2014).

[Opeb] *OpenStack administrator guides*. URL: http://docs.openstack.org/admin-guide-cloud/content/ (acedido em 18/06/2014).

[San86] R. Sandberg. *The Sun Network File System: Design, Implementation and Experience*. Rel. téc. in Proceedings of the Summer 1986 USENIX Technical Conference e Exhibition, 1986.

[She] *Sheepdog*. URL: https://github.com/sheepdog/sheepdog/ (acedido em 01/11/2015).

[Swi] *Swift Basics*. URL: https://julien.danjou.info/blog/2012/openstack-swift-consistency-analysis (acedido em 01/12/2015).

[Vmw] *Understanding Full Virtualization, Paravirtualization, and Hardware Assist*. White Paper. VMWare. URL: http://www.vmware.com/resources/techresources/1008 (acedido em 27/12/2015).

[Wei07a] A. W. e. B. S. A. e. M. C. Weil Sage A. e Leung. "RADOS: A Scalable, Reliable Storage Service for Petabyte-scale Storage Clusters". Em: *Proceedings of the 2Nd International Workshop on Petascale Data Storage: Held in Conjunction with Supercomputing '07*. PDSW '07. Reno, Nevada: ACM, 2007, pp. 35–44. ISBN: 978-1-59593-899-2. DOI: 10.1145/1374596.1374606. URL: http://doi.acm.org/10.1145/1374596.1374606.

[Wei07b] S. A. Weil. "Ceph: Reliable, Scalable, and High-performance Distributed Storage". AAI3288383. Tese de doutoramento. Santa Cruz, CA, USA, 2007. ISBN: 978-0-549-31235-2.

[Wei06a] S. A. e. M. E. L. e. L. D. D. E. e. M. C. Weil Sage A. e Brandt. "Ceph: A Scalable, High-performance Distributed File System". Em: *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*. OSDI '06. Seattle, Washington: USENIX Association, 2006, pp. 307–320. ISBN: 1-931971-47-1. URL: http://dl.acm.org/citation.cfm?id=1298455.1298485.

[Wei06b] S. A. e. M. E. L. e. M. C. Weil Sage A. e Brandt. "CRUSH: Controlled, Scalable, Decentralized Placement of Replicated Data". Em: *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*. SC '06. Tampa, Florida: ACM, 2006. ISBN: 0-7695-2700-0. DOI: 10.1145/1188455.1188582. URL: http://doi.acm.org/10.1145/1188455.1188582.

A

Caracterização do Subsistema de Armazenamento

Testes de Largura de Banda: Leituras Sequenciais

	Stripe unit size				
Block size	64 KB	128 KB	256 KB	512 KB	1024 KB
4 KB	102.68/ 91.83	92.59/ 89.53	93.58/ 89.25	94.25/ 90.28	92.74/ 89.39
16 KB	249.57/ 256.55	236.79/ 233.33	237.64/ 231.55	238.50/ 231.65	237.68/ 232.02
64 KB	362.86/ 359.90	354.28/ 352.28	353.03/ 350.43	364.18/ 345.24	340.31/ 335.06
256 KB	366.57/ 368.60	366.92/ 368.50	367.12/ 368.08	366.22/ 362.36	365.90/ 366.83
1024 KB	367.37/ 368.86	366.35/ 368.76	367.31/ 368.08	365.90/ 365.23	365.61/ 367.34
4096 KB	367.89/ 368.57	365.01/ 366.06	367.79/ 366.73	365.87/ 366.03	365.58/ 367.34
16384 KB	368.15/ 366.96	368.44/ 368.44	366.41/ 368.83	365.52/ 364.91	367.95/ 368.92

Tabela A.1: Taxa de transferência para leituras sequenciais num volume RAID-0, configurado com dois discos, apresentando na mesma tabela os resultados obtidos com discos de uma caixa e discos de duas caixas separadas.

	Stripe unit size				
Block size	64 KB	128 KB	256 KB	512 KB	1024 KB
4 KB	112.00	107.95	112.14	112.00	94.34
16 KB	266.44	265.68	266.88	266.85	238.42
64 KB	364.85	372.89	368.79	372.00	354.79
256 KB	532.54	525.80	406.58	406.94	393.72
1024 KB	527.12	531.53	529.45	528.92	410.40
4096 KB	532.34	527.85	530.79	528.58	515.90
16384 KB	533.63	524.75	530.86	528.18	525.73

Tabela A.2: Taxa de transferência para leituras sequenciais num volume RAID-0, configurado com três discos.

	Stripe unit size				
Block size	64 KB	128 KB	256 KB	512 KB	1024 KB
4 KB	98.13/ 91.30	107.08/ 105.24	90.97/ 104.59	90.38/ 98.13	90.63/ 105.22
16 KB	260.58/ 257.65	260.47/ 256.93	232.97/ 257.21	234.87/ 256.45	231.37/ 257.70
64 KB	365.10/ 365.68	368.37/ 363.43	353.26/ 367.05	354.28/ 359.96	356.02/ 364.94
256 KB	685.68/ 693.62	680.78/ 686.13	392.58/ 404.04	391.59/ 401.48	409.32/ 410.16
1024 KB	682.89/ 699.40	688.27/ 700.57	685.23/ 698.12	679.46/ 695.00	437.27/ 439.38
4096 KB	685.68/ 685.01	687.37/ 702.56	688.95/ 699.17	685.90/ 698.12	685.34/ 698.58
16384 KB	685.57/ 687.82	688.27/ 695.69	681.34/ 701.98	686.80/ 701.98	689.06/ 702.56

Tabela A.3: Taxa de transferência para leituras sequenciais num volume RAID-0, configurado com quatro discos, apresentando na mesma tabela os resultados obtidos com discos de uma caixa e discos de duas caixas separadas.

	Stripe unit size				
Block size	64 KB	128 KB	256 KB	512 KB	1024 KB
4 KB	96.54	96.59	105.56	103.40	111.00
16 KB	241.45	242.63	258.97	255.70	264.46
64 KB	356.11	359.41	361.92	362.67	365.90
256 KB	877.47	687.48	400.26	395.28	406.39
1024 KB	871.09	879.49	862.14	755.46	432.89
4096 KB	875.09	877.47	866.41	864.27	814.43
16384 KB	879.31	884.50	873.45	877.47	842.91

Tabela A.4: Taxa de transferência para leituras sequenciais num volume RAID-0, configurado com cinco discos.

	Stripe unit size				
Block size	64 KB	128 KB	256 KB	512 KB	1024 KB
4 KB	110.65/ 130.51	138.66/ 128.85	139.50/ 128.29	139.44/ 128.20	109.83/ 128.37
16 KB	265.29/ 298.06	315.67/ 299.02	304.09/ 295.12	308.18/ 295.39	264.32/ 294.01
64 KB	366.83/ 367.28	360.52/ 362.20	369.44/ 363.02	373.03/ 363.36	370.36/ 364.47
256 KB	1000.80/ 1038.90	715.75/ 738.56	409.84/ 408.40	414.17/ 406.82	412.87/ 404.97
1024 KB	1008.40/ 1022.80	1012.90/ 1035.10	1027.40/ 1034.90	655.16/ 678.14	431.87/ 439.89
4096 KB	987.48/ 994.62	1022.50/ 1038.20	1032.80/ 1033.40	1021.80/ 1017.40	828.10/ 883.38
16384 KB	1009.20/ 1040.40	1038.60/ 1045.10	1040.40/ 1038.30	1014.70/ 1024.80	983.19/ 1004.10

Tabela A.5: Taxa de transferência para leituras sequenciais num volume RAID-0, configurado com seis discos, apresentando na mesma tabela os resultados obtidos com discos de uma caixa e discos de duas caixas separadas.

	Stripe unit size				
Block size	64 KB	128 KB	256 KB	512 KB	1024 KB
4 KB	111.62	101.81	111.96	93.52	3.45
16 KB	265.36	236.74	265.77	236.79	9.31
64 KB	364.98	363.17	366.44	351.66	32.91
256 KB	1082.50	733.53	408.92	398.81	166.94
1024 KB	1140.10	1094.20	1146.80	760.25	80.78
4096 KB	1153.20	1110.40	1130.30	1096.70	897.95
16384 KB	1134.70	1082.20	1155.80	1066.40	1007.40

Tabela A.6: Taxa de transferência para leituras sequenciais num volume RAID-0, configurado com sete discos.

	Stripe unit size				
Block size	64 KB	128 KB	256 KB	512 KB	1024 KB
4 KB	111.59/ 89.83	111.42/ 99.40	111.44/ 89.88	119.15/ 106.46	102.72/ 76.91
16 KB	265.46/ 229.81	264.62/ 256.60	265.65/ 229.23	270.06/ 256.80	269.61/ 228.49
64 KB	364.56/ 347.84	365.17/ 360.92	367.50/ 348.91	352.20/ 359.87	349.50/ 348.48
256 KB	1135.30/ 1143.90	727.29/ 718.33	401.75/ 396.17	400.11/ 408.60	411.97/ 406.66
1024 KB	1262.70/ 1242.80	1232.30/ 1231.20	1276.90/ 1210.80	790.78/ 791.23	438.92/ 434.28
4096 KB	1290.10/ 1231.90	1323.50/ 1223.90	1295.40/ 1243.10	1322.20/ 1226.80	1069.80/ 1023.80
16384 KB	1266.20/ 1245.80	1329.90/ 1243.90	1314.60/ 1232.30	1299.90/ 1247.30	1318.80/ 1232.70

Tabela A.7: Taxa de transferência para leituras sequenciais num volume RAID-0, configurado com oito discos, apresentando na mesma tabela os resultados obtidos com discos de uma caixa e discos de duas caixas separadas.

	Stripe unit size				
Block size	64 KB	128 KB	256 KB	512 KB	1024 KB
4 KB	137.95	149.81	148.96	138.72	125.59
16 KB	322.17	320.37	319.71	307.48	320.47
64 KB	399.91	401.83	399.46	364.41	398.36
256 KB	1331.20	789.74	417.68	417.88	421.54
1024 KB	1447.90	1398.50	1414.40	699.40	441.23
4096 KB	1386.20	1498.20	1509.80	1463.40	1177.20
16384 KB	1525.60	1509.30	1511.50	1460.80	1436.80

Tabela A.8: Taxa de transferência para leituras sequenciais num volume RAID-0, configurado com nove discos.

	Stripe unit size				
Block size	64 KB	128 KB	256 KB	512 KB	1024 KB
4 KB	112.14/ 89.97	111.92/ 90.25	112.35/ 107.46	77.23/ 106.83	65.13/ 63.80
16 KB	264.21/ 227.11	264.89/ 229.74	267.31/ 258.19	234.55/ 254.93	236.93/ 230.10
64 KB	361.24/ 349.09	363.02/ 343.91	364.12/ 357.51	347.82/ 362.42	351.96/ 345.47
256 KB	1170.70/ 1091.10	730.33/ 679.35	408.20/ 408.84	393.35/ 400.64	395.58/ 406.86
1024 KB	1585.80/ 1609.50	1590.70/ 1569.10	1338.60/ 1316.70	779.90/ 777.30	436.04/ 433.34
4096 KB	1603.80/ 1594.50	1602.60/ 1640.40	1583.10/ 1617.80	1513.70/ 1577.90	1175.20/ 1062.90
16384 KB	1590.70/ 1640.40	1599.40/ 1647.70	1617.80/ 1635.90	1579.70/ 1611.40	1315.80/ 1425.80

Tabela A.9: Taxa de transferência para leituras sequenciais num volume RAID-0, configurado com dez discos, apresentando na mesma tabela os resultados obtidos com discos de uma caixa e discos de duas caixas separadas.

	Stripe unit size				
Block size	64 KB	128 KB	256 KB	512 KB	1024 KB
4 KB	138.44	138.33	149.06	149.23	123.40
16 KB	307.52	299.23	320.32	318.45	318.86
64 KB	363.68	373.62	401.60	397.83	398.47
256 KB	1238.10	728.81	408.96	422.90	400.03
1024 KB	1748.20	1805.20	1480.90	678.58	439.93
4096 KB	1821.30	1801.30	1643.70	1721.80	1291.40
16384 KB	1809.20	1835.10	1672.60	1823.80	1537.60

Tabela A.10: Taxa de transferência para leituras sequenciais num volume RAID-0, configurado com onze discos.

	Stripe unit size				
Block size	64 KB	128 KB	256 KB	512 KB	1024 KB
4 KB	117.93/ 126.99	112.24/ 126.03	111.56/ 128.88	88.43/ 97.79	70.24/ 68.17
16 KB	266.90/ 290.38	264.66/ 284.59	263.63/ 291.88	261.31/ 291.88	191.56/ 289.64
64 KB	352.23/ 362.73	359.62/ 361.77	356.63/ 365.61	365.13/ 353.89	368.60/ 361.55
256 KB	1052.10/ 1208.70	731.73/ 742.62	409.24/ 396.62	404.39/ 406.62	400.56/ 407.21
1024 KB	1876.40/ 1943.80	1858.50/ 1831.30	1338.60/ 1463.40	757.09/ 486.13	423.97/ 435.36
4096 KB	1892.90/ 1922.20	1867.80/ 1767.50	1840.10/ 1938.50	1682.20/ 1831.30	1196.10/ 1281.70
16384 KB	1886.80/ 1994.20	1899.90/ 2006.90	1864.40/ 1963.60	1873.80/ 1944.10	1462.40/ 1571.20

Tabela A.11: Taxa de transferência para leituras sequenciais num volume RAID-0, configurado com doze discos, apresentando na mesma tabela os resultados obtidos com discos de uma caixa e discos de duas caixas separadas.

	Stripe unit size				
Block size	64 KB	128 KB	256 KB	512 KB	1024 KB
4 KB	57.18	15.97	25.73	1.46	53.52
16 KB	256.47	19.79	33.59	4.82	70.48
64 KB	360.52	329.04	323.26	88.27	145.20
256 KB	1154.50	663.76	377.39	63.88	155.95
1024 KB	2727.40	2126.80	1296.70	135.47	161.82
4096 KB	3300.60	3077.40	2882.50	1706.70	1140.40
16384 KB	3515.90	3371.20	3515.90	3416.20	2615.60

Tabela A.12: Taxa de transferência para leituras sequenciais num volume RAID-0, configurado com vinte e quatro discos, apresentando na mesma tabela os resultados obtidos com discos de uma caixa e discos de duas caixas separadas.

	Stripe unit size				
Block size	64 KB	128 KB	256 KB	512 KB	1024 KB
4 KB	4.81	16.51	1.61	35.56	67.39
16 KB	17.64	20.59	9.32	40.77	83.71
64 KB	161.49	347.35	51.75	138.28	144.99
256 KB	1190.50	723.41	69.68	152.28	164.12
1024 KB	3165.40	2576.20	804.28	120.96	163.70
4096 KB	3912.20	3771.70	3263.80	1983.60	1030.10
16384 KB	2717.20	3831.70	3596.20	3976.80	3537.20

Tabela A.13: Taxa de transferência para leituras sequenciais num volume RAID-0, configurado com trinta e dois discos, apresentando na mesma tabela os resultados obtidos com discos de uma caixa e discos de duas caixas separadas.

	Stripe unit size
Block size	64 KB
4 KB	110.66
16 KB	255.66
64 KB	341.56
256 KB	360.12
1024 KB	360.86
4096 KB	361.86
16384 KB	361.30

Tabela A.14: Taxa de transferência para leituras sequenciais num volume RAID-5, configurado com três discos.

	Stripe unit size
Block size	64 KB
4 KB	102.03
16 KB	238.88
64 KB	323.61
256 KB	362.99
1024 KB	364.31
4096 KB	362.48
16384 KB	364.25

Tabela A.15: Taxa de transferência para leituras sequenciais num volume RAID-6, configurado com quatro discos, dois dos quais configurados como discos de paridade.

	Stripe unit size
Block size	64 KB
4 KB	84.07
16 KB	221.14
64 KB	341.31
256 KB	369.216
1024 KB	363.87
4096 KB	350.90
16384 KB	359

Tabela A.16: Taxa de transferência para leituras sequenciais num volume RAID-10, configurado com quatro discos, dois dos quais configurados como discos de paridade.

Testes de Largura de Banda: Escritas Sequenciais

	Stripe unit size				
Block size	64 KB	128 KB	256 KB	512 KB	1024 KB
4 KB	0.47/ 0.47	0.47/ 0.32	0.47/ 0.47	0.48/ 0.48	0.48/ 0.48
16 KB	2.14/ 2.14	1.99/ 0.74	1.84/ 1.87	1.88/ 1.87	1.86/ 1.86
64 KB	13.75/ 13.48	9.47/ 3.54	8.27/ 8.33	7.74/ 7.77	7.00/ 7.10
256 KB	28.15/ 28.11	28.11/ 28.01	44.34/ 45.66	33.07/ 33.97	28.85/ 29.53
1024 Kb	91.57/ 91.04	90.35/ 90.70	52.55/ 52.37	89.93/ 90.54	102.81/ 105.47
4096 KB	206.40/ 208.01	204.03/ 206.47	147.54/ 146.49	97.79/ 111.94	115.54/ 165.18
16384 KB	303.71/ 291.68	297.19/ 306.27	263.81/ 263.46	126.86/ 130.36	136.82/ 135.83

Tabela A.17: Taxa de transferência para escritas sequenciais num volume RAID-0, configurado com dois discos, apresentando na mesma tabela os resultados obtidos com discos de uma caixa e discos de duas caixas separadas.

	Stripe unit size				
Block size	64 KB	128 KB	256 KB	512 KB	1024 KB
4 KB	0.47	0.47	0.47	0.48	0.48
16 KB	2.08	1.77	1.86	1.84	1.86
64 KB	13.31	9.50	8.14	6.68	7.04
256 KB	28.61	33.69	43.83	32.08	27.13
1024 KB	98.28	97.88	64.60	94.97	99.90
4096 KB	250.02	252.85	178.18	135.83	236.25
16384 KB	386.71	360.30	348.42	184.30	193.85

Tabela A.18: Taxa de transferência para escritas sequenciais num volume RAID-0, configurado com três discos.

	Stripe unit size				
Block size	64 KB	128 KB	256 KB	512 KB	1024 KB
4 KB	0.46/ 0.48	0.47/ 0.48	0.47/ 0.47	0.48/ 0.48	0.48/ 0.48
16 KB	2.06/ 2.07	1.81/ 1.86	1.79/ 1.92	1.86/ 1.86	1.88/ 1.87
64 KB	13.36/ 13.95	9.36/ 9.41	7.74/ 7.90	6.92/ 6.98	7.13/ 6.97
256 KB	29.01/ 29.10	39.21/ 37.85	44.68/ 42.91	30.49/ 32.01	23.52/ 21.27
1024 KB	103.24/ 102.99	103.14/ 102.66	71.56/ 63.54	90.04/ 95.88	85.76/ 92.60
4096 KB	269.09/ 275.56	274.39/ 275.45	175.12/ 182.10	163.18/ 174.26	208.03/ 260.08
16384 KB	494.26/ 511.25	479.29/ 456.35	382.66/ 415.81	184.28/ 231.78	259.93/ 286.38

Tabela A.19: Taxa de transferência para escritas sequenciais num volume RAID-0, configurado com quatro discos, apresentando na mesma tabela os resultados obtidos com discos de uma caixa e discos de duas caixas separadas.

	Stripe unit size				
Block size	64 KB	128 KB	256 KB	512 KB	1024 KB
4 KB	0.46	0.47	0.47	0.47	0.48
16 KB	1.72	1.74	1.78	1.85	1.86
64 KB	13.20	8.68	6.43	6.75	6.83
256 KB	29.30	37.90	41.88	27.25	22.19
1024 KB	104.89	100.95	69.38	67.15	77.32
4096 KB	287.77	296.58	213.64	106.61	204.71
16384 KB	568.18	509.51	385.40	178.92	286.57

Tabela A.20: Taxa de transferência para escritas sequenciais num volume RAID-0, configurado com cinco discos.

	Stripe unit size				
Block size	64 KB	128 KB	256 KB	512 KB	1024 KB
4 KB	0.47/ 0.49	0.47/ 0.48	0.47/ 0.47	0.48/ 0.48	0.48/ 0.48
16 KB	1.79/ 1.85	1.77/ 1.93	1.86/ 1.80	1.86/ 1.86	1.82/ 1.86
64 KB	13.00/ 12.57	8.28/ 8.59	7.02/ 6.69	6.83/ 6.95	3.01/ 6.93
256 KB	30.76/ 31.19	38.45/ 38.23	37.07/ 38.41	29.19/ 29.74	13.04/ 22.75
1024 KB	104.30/ 107.84	104.62/ 106.62	72.99/ 71.05	75.65/ 80.68	38.31/ 55.76
4096 KB	291.92/ 295.98	318.30/ 310.69	225.88/ 221.17	161.90/ 139.17	107.73/ 158.51
16384 KB	588.26/ 604.37	522.52/ 561.26	490.85/ 476.25	257.49/ 221.04	259.66/ 237.15

Tabela A.21: Taxa de transferência para escritas sequenciais num volume RAID-0, configurado com seis discos, apresentando na mesma tabela os resultados obtidos com discos de uma caixa e discos de duas caixas separadas.

	Stripe unit size				
Block size	64 KB	128 KB	256 KB	512 KB	1024 KB
4 KB	0.46	0.47	0.47	0.48	0.48
16 KB	1.71	1.75	1.84	1.86	0.76
64 KB	13.14	8.36	6.79	6.90	2.87
256 KB	31.14	36.99	35.88	25.18	12.36
1024 KB	108.93	102.73	66.68	55.54	35.60
4096 KB	305.75	317.61	218.18	115.02	145.64
16384 KB	672.06	578.68	451.44	164.64	285.62

Tabela A.22: Taxa de transferência para escritas sequenciais num volume RAID-0, configurado com sete discos.

	Stripe unit size				
Block size	64 KB	128 KB	256 KB	512 KB	1024 KB
4 KB	0.47/ 0.45	0.47/ 0.47	0.47/ 0.47	0.48/ 0.48	0.48/ 0.48
16 KB	1.83/ 1.62	1.83/ 1.72	1.83/ 1.81	1.87/ 1.88	1.88/ 1.89
64 KB	12.76/ 12.11	7.18/ 6.57	6.83/ 6.61	7.00/ 7.24	7.24/ 7.29
256 KB	31.30/ 30.82	35.51/ 34.07	32.29/ 28.37	23.82/ 24.17	25.29/ 25.14
1024 KB	109.37/ 104.43	105.68/ 101.18	67.77/ 65.19	59.58/ 66.43	60.38/ 61.94
4096 KB	299.70/ 327.94	301.53/ 307.25	189.26/ 185.16	108.81/ 105.59	160.10/ 179.30
16384 KB	544.86/ 597.48	490.10/ 546.77	453.00/ 463.97	201.57/ 178.11	240.65/ 249.59

Tabela A.23: Taxa de transferência para escritas sequenciais num volume RAID-0, configurado com oito discos, apresentando na mesma tabela os resultados obtidos com discos de uma caixa e discos de duas caixas separadas.

	Stripe unit size				
Block size	64 KB	128 KB	256 KB	512 KB	1024 KB
4 KB	0.47	0.47	0.48	0.48	0.48
16 KB	1.86	1.75	1.84	1.86	1.89
64 KB	12.35	6.21	6.63	6.92	7.27
256 KB	31.49	33.41	25.60	23.10	25.93
1024 KB	105.28	99.22	63.67	51.12	63.28
4096 KB	326.74	311.08	213.37	88.43	138.33
16384 KB	640.25	779.03	558.57	163.24	254.23

Tabela A.24: Taxa de transferência para escritas sequenciais num volume RAID-0, configurado com nove discos.

	Stripe unit size				
Block size	64 KB	128 KB	256 KB	512 KB	1024 KB
4 KB	0.45/ 0.47	0.47/ 0.48	0.47/ 0.49	0.48/ 0.49	0.48/ 0.49
16 KB	1.56/ 1.62	1.79/ 1.75	1.85/ 1.87	1.89/ 1.85	1.89/ 1.87
64 KB	8.42/ 8.06	6.47/ 6.19	6.78/ 7.11	7.39/ 6.74	7.30/ 7.07
256 KB	31.97/ 31.43	33.25/ 34.30	24.86/ 25.68	25.01/ 22.26	26.57/ 23.83
1024 KB	105.00/ 102.82	107.58/ 100.71	56.57/ 53.41	56.36/ 50.37	63.20/ 56.39
4096 KB	339.81/ 330.16	313.26/ 302.88	210.71/ 199.86	122.37/ 94.78	127.37/ 109.56
16384 KB	637.53/ 496.72	542.81/ 556.27	503.70/ 444.08	185.79/ 171.04	222.86/ 203.41

Tabela A.25: Taxa de transferência para escritas sequenciais num volume RAID-0, configurado com dez discos, apresentando na mesma tabela os resultados obtidos com discos de uma caixa e discos de duas caixas separadas.

	Stripe unit size				
Block size	64 KB	128 KB	256 KB	512 KB	1024 KB
4 KB	0.48	0.47	0.48	0.48	0.48
16 KB	1.86	1.84	1.88	1.86	1.87
64 KB	8.39	6.71	7.22	6.89	7.10
256 KB	31.79	32.56	26.91	22.96	23.95
1024 KB	106.52	104.67	56.66	50.26	59.74
4096 KB	321.82	311.06	204.04	110.40	118.06
16384 KB	663.66	814.11	488.33	187.99	257.56

Tabela A.26: Taxa de transferência para escritas sequenciais num volume RAID-0, configurado com onze discos.

	Stripe unit size				
Block size	64 KB	128 KB	256 KB	512 KB	1024 KB
4 KB	0.47/ 0.46	0.47/ 0.47	0.47/ 0.48	0.48/ 0.48	0.48/ 0.16
16 KB	1.78/ 1.81	1.83/ 1.80	1.82/ 1.89	1.88/ 1.85	1.88/ 0.93
64 KB	7.09/ 7.11	6.80/ 6.59	6.44/ 7.34	7.26/ 6.83	7.29/ 7.01
256 KB	31.53/ 31.52	31.69/ 32.21	21.33/ 25.19	25.45/ 22.79	26.32/ 22.77
1024 KB	104.02/ 98.71	102.25/ 100.80	54.66/ 62.48	52.65/ 50.35	63.69/ 60.33
4096 KB	356.29/ 333.33	334.34/ 297.22	177.62/ 189.50	98.90/ 94.39	142.47/ 140.10
16384 KB	906.29/ 661.35	543.23/ 666.93	487.14/ 526.59	182.35/ 192.24	254.31/ 306.83

Tabela A.27: Taxa de transferência para escritas sequenciais num volume RAID-0, configurado com doze discos, apresentando na mesma tabela os resultados obtidos com discos de uma caixa e discos de duas caixas separadas.

	Stripe unit size				
Block size	64 KB	128 KB	256 KB	512 KB	1024 KB
4 KB	0.46	0.47	0.47	0.40	0.48
16 KB	1.72	1.79	1.84	0.71	1.87
64 KB	6.17	5.95	6.65	2.72	7.08
256 KB	19.76	17.53	20.39	13.18	24.10
1024 KB	88.95	82.63	45.14	55.71	64.47
4096 KB	311.94	254.77	170.42	108.10	123.63
16384 KB	630.72	575.35	500.99	198.91	218.97

Tabela A.28: Taxa de transferência para escritas sequenciais num volume RAID-0, configurado com vinte e quatro discos, apresentando na mesma tabela os resultados obtidos com discos de uma caixa e discos de duas caixas separadas.

	Stripe unit size				
Block size	64 KB	128 KB	256 KB	512 KB	1024 KB
4 KB	0.46	0.47	0.48	0.48	0.48
16 KB	1.64	1.85	0.77	1.87	1.88
64 KB	4.60	7.06	3.13	7.03	7.22
256 KB	13.13	21.70	13.33	23.68	25.79
1024 KB	62.25	75.27	52.38	56.11	77.27
4096 KB	219.80	258.45	112.36	120.70	148.58
16384 KB	546.56	740.39	263.89	228.17	248.85

Tabela A.29: Taxa de transferência para escritas sequenciais num volume RAID-0, configurado com trinta e dois discos, apresentando na mesma tabela os resultados obtidos com discos de uma caixa e discos de duas caixas separadas.

	Stripe unit size
Block size	64 KB
4 KB	0.46
16 KB	1.63
64 KB	5.25
256 KB	28.00
1024 KB	29.35
4096 KB	77.70
16384 KB	156.58

Tabela A.30: Taxa de transferência para escritas sequenciais num volume RAID-5, configurado com três discos.

	Stripe unit size
Block size	64 KB
4 KB	0.42
16 KB	1.37
64 KB	3.34
256 KB	27.64
1024 KB	24.76
4096 KB	72.28
16384 KB	134

Tabela A.31: Taxa de transferência para escritas sequenciais num volume RAID-6, configurado com quatro discos, dois dos quais são discos de paridade.

	Stripe unit size
Block size	64 KB
4 KB	0.27
16 KB	2.02
64 KB	10.53
256 KB	27.81
1024 KB	88.79
4096 KB	195.392
16384 KB	237.758

Tabela A.32: Taxa de transferência para escritas sequenciais num volume RAID-10, configurado com quatro discos, dois dos quais são discos de paridade.

Testes de IOPS tests: leituras aleatórias

	Stripe unit size				
Block size	64 KB	128 KB	256 KB	512 KB	1024 KB
4 KB	198/ 196	196/ 123	198/ 197	198/ 195	198/ 197
16 KB	198/ 196	199/ 198	200/ 199	199/ 198	201/ 200
64 KB	190/ 189	191/ 189	192/ 190	193/ 192	194/ 194
256 KB	145/ 142	149/ 145	161/ 161	162/ 160	163/ 163
1024 KB	101/ 100	102/ 103	104/ 105	104/ 104	93/ 93
4096 KB	51/ 50	51/ 52	51/ 53	53/ 54	53/ 54
16384 KB	18/ 18	16/ 18	18/ 18	19/ 18	17/ 18

Tabela A.33: IOPS para leituras aleatórias num volume RAID-0, configurado com dois discos, apresentando na mesma tabela os resultados obtidos com discos de uma caixa e discos de duas caixas separadas.

	Stripe unit size				
Block size	64 KB	128 KB	256 KB	512 KB	1024 KB
4 KB	201	202	202	201	202
16 KB	201	203	205	204	205
64 KB	191	192	195	196	200
256 KB	143	146	158	161	166
1024 KB	112	112	111	105	93
4096 KB	63	63	62	61	59
16384 KB	24	25	25	25	25

Tabela A.34: IOPS para leituras aleatórias num volume RAID-0, configurado com dois discos.

	Stripe unit size				
Block size	64 KB	128 KB	256 KB	512 KB	1024 KB
4 KB	202/ 203	204/ 205	204/ 209	205/ 205	208/ 203
16 KB	202/ 205	207/ 207	204/ 211	207/ 207	211/ 207
64 KB	191/ 194	195/ 196	196/ 202	201/ 201	204/ 199
256 KB	141/ 140	152/ 152	156/ 163	163/ 162	172/ 167
1024 KB	120/ 119	119/ 118	119/ 118	109/ 112	96/ 92
4096 KB	71/ 71	72/ 65	72/ 72	71/ 69	71/ 71
16384 KB	26/ 30	28/ 32	31/ 31	32/ 32	32/ 32

Tabela A.35: IOPS para leituras aleatórias num volume RAID-0, configurado com quatro discos, apresentando na mesma tabela os resultados obtidos com discos de uma caixa e discos de duas caixas separadas.

	Stripe unit size				
Block size	64 KB	128 KB	256 KB	512 KB	1024 KB
4 KB	201	201	203	203	200
16 KB	200	202	206	207	203
64 KB	186	189	195	199	200
256 KB	135	145	153	159	163
1024 KB	122	122	117	106	89
4096 KB	76	76	77	73	71
16384 KB	35	37	37	37	33

Tabela A.36: IOPS para leituras aleatórias num volume RAID-0, configurado com cinco discos.

	Stripe unit size				
Block size	64 KB	128 KB	256 KB	512 KB	1024 KB
4 KB	207/ 217	207/ 215	210/ 213	210/ 208	137/ 202
16 KB	206/ 216	210/ 218	214/ 216	213/ 210	212/ 204
64 KB	193/ 201	197/ 204	204/ 205	207/ 204	209/ 201
256 KB	136/ 136	150/ 153	157/ 160	166/ 163	174/ 168
1024 KB	126/ 126	126/ 126	120/ 119	111/ 109	92/ 89
4096 KB	82/ 82	84/ 85	82/ 83	77/ 79	73/ 74
16384 KB	35/ 34	41/ 39	39/ 41	41/ 41	36/ 37

Tabela A.37: IOPS para leituras aleatórias num volume RAID-0, configurado com seis discos, apresentando na mesma tabela os resultados obtidos com discos de uma caixa e discos de duas caixas separadas.

	Stripe unit size				
Block size	64 KB	128 KB	256 KB	512 KB	1024 KB
4 KB	208	208	209	212	216
16 KB	206	210	212	219	220
64 KB	192	195	201	216	219
256 KB	135	143	152	165	177
1024 KB	125	127	116	101	94
4096 KB	85	85	83	79	69
16384 KB	42	42	43	40	38

Tabela A.38: IOPS para leituras aleatórias num volume RAID-0, configurado com sete discos.

	Stripe unit size				
Block size	64 KB	128 KB	256 KB	512 KB	1024 KB
4 KB	221/ 203	216/ 209	211/ 204	222/ 223	227/ 230
16 KB	220/ 201	219/ 213	212/ 207	225/ 226	231/ 234
64 KB	204/ 186	202/ 197	204/ 197	218/ 220	228/ 229
256 KB	147/ 142	149/ 149	153/ 146	169/ 170	186/ 187
1024 KB	129/ 130	131/ 132	128/ 126	106/ 104	95/ 97
4096 KB	89/ 84	88/ 89	90/ 88	87/ 85	78/ 76
16384 KB	47/ 47	48/ 48	48/ 47	47/ 46	49/ 46

Tabela A.39: IOPS para leituras aleatórias num volume RAID-0, configurado com oito discos, apresentando na mesma tabela os resultados obtidos com discos de uma caixa e discos de duas caixas separadas.

	Stripe unit size				
Block size	64 KB	128 KB	256 KB	512 KB	1024 KB
4 KB	220	208	216	216	236
16 KB	217	213	219	219	240
64 KB	201	199	209	217	242
256 KB	137	140	155	165	197
1024 KB	133	127	116	95	98
4096 KB	95	96	95	83	75
16384 KB	45	52	53	50	50

Tabela A.40: IOPS para leituras aleatórias num volume RAID-0, configurado com nove discos.

	Stripe unit size				
Block size	64 KB	128 KB	256 KB	512 KB	1024 KB
4 KB	207/ 202	217/ 204	224/ 228	238/ 208	240/ 222
16 KB	205/ 200	220/ 206	229/ 232	241/ 211	244/ 224
64 KB	186/ 182	202/ 189	218/ 222	237/ 206	242/ 223
256 KB	135/ 135	150/ 142	160/ 164	182/ 159	197/ 185
1024 KB	136/ 135	126/ 126	117/ 120	106/ 98	98/ 91
4096 KB	95/ 97	90/ 90	90/ 93	84/ 84	72/ 71
16384 KB	54/ 53	52/ 48	51/ 52	47/ 48	48/ 48

Tabela A.41: IOPS para leituras aleatórias num volume RAID-0, configurado com dez discos, apresentando na mesma tabela os resultados obtidos com discos de uma caixa e discos de duas caixas separadas.

	Stripe unit size				
Block size	64 KB	128 KB	256 KB	512 KB	1024 KB
4 KB	232	237	232	216	227
16 KB	229	238	235	219	231
64 KB	211	222	223	218	236
256 KB	135	151	167	164	186
1024 KB	139	126	121	92	91
4096 KB	98	98	95	82	72
16384 KB	52	56	53	54	52

Tabela A.42: IOPS para leituras aleatórias num volume RAID-0, configurado com onze discos.

	Stripe unit size				
Block size	64 KB	128 KB	256 KB	512 KB	1024 KB
4 KB	224/ 233	222/ 216	220/ 233	246/ 213	250/ 211
16 KB	229/ 231	224/ 219	224/ 236	249/ 216	254/ 219
64 KB	207/ 212	206/ 201	214/ 225	247/ 215	252/ 224
256 KB	152/ 154	152/ 152	155/ 166	185/ 162	207/ 180
1024 KB	139/ 140	130/ 129	126/ 129	104/ 97	98/ 86
4096 KB	102/ 106	100/ 101	92/ 100	89/ 89	74/ 75
16384 KB	56/ 56	58/ 58	57/ 59	54/ 54	51/ 53

Tabela A.43: IOPS para leituras aleatórias num volume RAID-0, configurado com doze discos, apresentando na mesma tabela os resultados obtidos com discos de uma caixa e discos de duas caixas separadas.

	Stripe unit size				
Block size	64 KB	128 KB	256 KB	512 KB	1024 KB
4 KB	261	265	266	253	304
16 KB	255	274	277	269	318
64 KB	220	246	261	266	322
256 KB	168	170	166	175	230
1024 KB	147	159	121	95	100
4096 KB	117	114	111	90	72
16384 KB	75	75	75	68	60

Tabela A.44: IOPS para leituras aleatórias num volume RAID-0, configurado com vinte e quatro discos, apresentando na mesma tabela os resultados obtidos com discos de uma caixa e discos de duas caixas separadas.

	Stripe unit size				
Block size	64 KB	128 KB	256 KB	512 KB	1024 KB
4 KB	341	339	338	281	368
16 KB	342	347	363	300	389
64 KB	287	302	339	309	398
256 KB	194	204	207	197	260
1024 KB	177	166	123	93	102
4096 KB	131	138	134	81	73
16384 KB	90	86	90	79	68

Tabela A.45: IOPS para leituras aleatórias num volume RAID-0, configurado com trinta e dois discos, apresentando na mesma tabela os resultados obtidos com discos de uma caixa e discos de duas caixas separadas.

	Stripe unit size
Block size	64 KB
4 KB	197
16 KB	196
64 KB	186
256 KB	138
1024 KB	99
4096 KB	50
16384 KB	18

Tabela A.46: Taxa de transferência para leituras aleatórias num volume RAID-5, configurado com três discos.

	Stripe unit size
Block size	64 KB
4 KB	197
16 KB	196
64 KB	186
256 KB	138
1024 KB	99
4096 KB	50
16384 KB	18

Tabela A.47: IOPS para leituras aleatórias num volume RAID-6, configurado com quatro discos, dois dos quais sãos discos de paridade.

	Stripe unit size
Block size	64 KB
4 KB	196
16 KB	196
64 KB	189
256 KB	145
1024 KB	104
4096 KB	50
16384 KB	16

Tabela A.48: IOPS para leituras aleatórias num volume RAID-10, configurado com quatro discos, dois dos quais sãos discos de paridade.

Testes de IOPS: Escrita Aleatória

	Stripe unit size				
Block size	64 KB	128 KB	256 KB	512 KB	1024 KB
4 KB	117/ 122	132/ 131	112/ 112	110/ 114	110/ 112
16 KB	107/ 115	149/ 150	102/ 107	114/ 113	97/ 100
64 KB	110/ 105	144/ 150	94/ 101	92/ 105	99/ 106
256 KB	92/ 100	118/ 117	74/ 83	104/ 105	76/ 77
1024 KB	92/ 43	92/ 91	56/ 55	61/ 53	47/ 46
4096 KB	29/ 43	38/ 48	35/ 34	21/ 27	33/ 35
16384 KB	12/ 12	15/ 16	14/ 12	7/ 7	8/ 8

Tabela A.49: IOPS para escritas aleatórias num volume RAID-0, configurado com dois discos, apresentando na mesma tabela os resultados obtidos com discos de uma caixa e discos de duas caixas separadas.

	Stripe unit size				
Block size	64 KB	128 KB	256 KB	512 KB	1024 KB
4 KB	109	109	110	108	109
16 KB	103	104	104	102	103
64 KB	92	93	92	94	94
256 KB	81	57	83	78	81
1024 KB	68	80	44	37	48
4096 KB	30	30	24	18	44
16384 KB	13	12	10	7	8

Tabela A.50: IOPS para escritas aleatórias num volume RAID-0, configurado com três discos.

	Stripe unit size				
Block size	64 KB	128 KB	256 KB	512 KB	1024 KB
4 KB	112/ 110	109/ 108	109/ 115	109/ 109	112/ 104
16 KB	102/ 104	104/ 102	104/ 110	102/ 103	107/ 99
64 KB	95/ 96	100/ 93	96/ 100	101/ 99	102/ 93
256 KB	38/ 56	59/ 58	79/ 85	78/ 81	81/ 74
1024 KB	34/ 62	44/ 37	29/ 43	39/ 40	52/ 48
4096 KB	21/ 41	25/ 20	20/ 25	15/ 15	18/ 18
16384 KB	11/ 12	13/ 11	11/ 11	8/ 7	10/ 9

Tabela A.51: IOPS para escritas aleatórias num volume RAID-0, configurado com quatro discos, apresentando na mesma tabela os resultados obtidos com discos de uma caixa e discos de duas caixas separadas.

	Stripe unit size				
Block size	64 KB	128 KB	256 KB	512 KB	1024 KB
4 KB	106	105	104	105	103
16 KB	101	101	102	101	98
64 KB	95	94	96	95	94
256 KB	41	63	81	81	77
1024 KB	31	31	31	43	48
4096 KB	18	19	20	17	20
16384 KB	10	10	10	8	10

Tabela A.52: IOPS para escritas aleatórias num volume RAID-0, configurado com cinco discos.

	Stripe unit size				
Block size	64 KB	128 KB	256 KB	512 KB	1024 KB
4 KB	107/ 117	106/ 116	108/ 113	106/ 107	114/ 102
16 KB	102/ 112	103/ 111	104/ 107	104/ 104	109/ 98
64 KB	97/ 105	99/ 104	98/ 101	99/ 98	102/ 94
256 KB	42/ 42	63/ 65	83/ 85	82/ 82	88/ 79
1024 KB	29/ 30	28/ 29	33/ 33	43/ 43	51/ 50
4096 KB	18/ 20	20/ 19	26/ 19	16/ 16	23/ 22
16384 KB	11/ 11	12/ 12	11/ 12	9/ 8	10/ 11

Tabela A.53: IOPS para escritas aleatórias num volume RAID-0, configurado com seis discos, apresentando na mesma tabela os resultados obtidos com discos de uma caixa e discos de duas caixas separadas.

	Stripe unit size				
Block size	64 KB	128 KB	256 KB	512 KB	1024 KB
4 KB	102	103	105	105	43
16 KB	101	101	101	102	81
64 KB	96	96	96	97	100
256 KB	44	63	80	83	84
1024 KB	28	26	34	42	52
4096 KB	17	18	20	18	22
16384 KB	9	11	10	8	10

Tabela A.54: IOPS para escritas aleatórias num volume RAID-0, configurado com sete discos.

	Stripe unit size				
Block size	64 KB	128 KB	256 KB	512 KB	1024 KB
4 KB	116/ 103	111/ 104	104/ 102	108/ 113	113/ 118
16 KB	111/ 100	107/ 105	103/ 101	108/ 113	109/ 114
64 KB	103/ 93	99/ 98	98/ 95	104/ 107	106/ 110
256 KB	46/ 48	66/ 67	81/ 81	86/ 89	90/ 92
1024 KB	25/ 26	25/ 27	35/ 36	45/ 46	53/ 57
4096 KB	19/ 17	18/ 18	18/ 19	19/ 20	25/ 26
16384 KB	9/ 9	11/ 11	11/ 11	9/ 9	11/ 11

Tabela A.55: IOPS para escritas aleatórias num volume RAID-0, configurado com oito discos, apresentando na mesma tabela os resultados obtidos com discos de uma caixa e discos de duas caixas separadas.

	Stripe unit size				
Block size	64 KB	128 KB	256 KB	512 KB	1024 KB
4 KB	109	101	106	105	117
16 KB	105	98	103	103	114
64 KB	100	93	98	99	109
256 KB	46	61	84	85	92
1024 KB	26	27	35	44	57
4096 KB	17	19	20	19	22
16384 KB	12	12	13	10	13

Tabela A.56: IOPS para escritas aleatórias num volume RAID-0, configurado com nove discos.

	Stripe unit size				
Block size	64 KB	128 KB	256 KB	512 KB	1024 KB
4 KB	99/ 98	113/ 100	111/ 111	119/ 100	120/ 106
16 KB	98/ 95	109/ 97	109/ 107	116/ 97	117/ 104
64 KB	96/ 89	103/ 92	103/ 102	111/ 92	111/ 98
256 KB	47/ 43	66/ 60	89/ 87	95/ 78	95/ 86
1024 KB	24/ 22	28/ 26	36/ 35	49/ 45	58/ 53
4096 KB	18/ 17	20/ 18	19/ 19	20/ 19	25/ 23
16384 KB	9/ 9	11/ 10	11/ 11	9/ 9	11/ 11

Tabela A.57: IOPS para escritas aleatórias num volume RAID-0, configurado com dez discos, apresentando na mesma tabela os resultados obtidos com discos de uma caixa e discos de duas caixas separadas.

	Stripe unit size				
Block size	64 KB	128 KB	256 KB	512 KB	1024 KB
4 KB	115	111	114	104	110
16 KB	112	112	111	102	107
64 KB	105	108	105	96	102
256 KB	47	67	88	83	87
1024 KB	23	27	37	45	57
4096 KB	18	19	20	20	24
16384 KB	11	13	11	10	12

Tabela A.58: IOPS para escritas aleatórias num volume RAID-0, configurado com onze discos.

	Stripe unit size				
Block size	64 KB	128 KB	256 KB	512 KB	1024 KB
4 KB	110/ 114	110/ 108	104/ 117	121/ 105	123/ 104
16 KB	106/ 113	106/ 104	102/ 111	117/ 101	120/ 100
64 KB	101/ 107	101/ 100	100/ 107	113/ 97	112/ 96
256 KB	49/ 49	66/ 65	86/ 90	97/ 84	98/ 84
1024 KB	23/ 23	28/ 30	38/ 39	50/ 48	61/ 52
4096 KB	19/ 20	19/ 19	18/ 19	21/ 22	24/ 25
16384 KB	12/ 11	11/ 12	11/ 12	10/ 11	11/ 12

Tabela A.59: IOPS para escritas aleatórias num volume RAID-0, configurado com doze discos, apresentando na mesma tabela os resultados obtidos com discos de uma caixa e discos de duas caixas separadas.

	Stripe unit size				
Block size	64 KB	128 KB	256 KB	512 KB	1024 KB
4 KB	107	106	108	53	120
16 KB	107	103	108	102	125
64 KB	104	102	104	97	117
256 KB	54	72	93	88	104
1024 KB	22	32	44	53	65
4096 KB	16	16	19	24	30
16384 KB	9	10	11	10	11

Tabela A.60: IOPS para escritas aleatórias num volume RAID-0, configurado com vinte e quatro discos, apresentando na mesma tabela os resultados obtidos com discos de uma caixa e discos de duas caixas separadas.

	Stripe unit size				
Block size	64 KB	128 KB	256 KB	512 KB	1024 KB
4 KB	101	123	49	108	133
16 KB	106	118	118	102	131
64 KB	113	111	111	98	122
256 KB	58	81	97	86	104
1024 KB	24	36	47	53	67
4096 KB	15	14	20	27	33
16384 KB	10	10	12	12	13

Tabela A.61: IOPS para escritas aleatórias num volume RAID-0, configurado com trinta e dois discos, apresentando na mesma tabela os resultados obtidos com discos de uma caixa e discos de duas caixas separadas.

	Stripe unit size
Block size	64 KB
4 KB	68
16 KB	65
64 KB	63
256 KB	46
1024 KB	20
4096 KB	14
16384 KB	5

Tabela A.62: IOPS para escritas aleatórias num volume RAID-5, configurado com três discos.

	Stripe unit size
Block size	64 KB
4 KB	42
16 KB	39
64 KB	49
256 KB	70
1024 KB	13
4096 KB	9
16384 KB	3

Tabela A.63: IOPS para escritas aleatórias num volume RAID-6, configurado com quatro discos, dois dos quais como discos de paridade.

	Stripe unit size
Block size	64 KB
4 KB	75
16 KB	67
64 KB	65
256 KB	58
1024 KB	56
4096 KB	22
16384 KB	7

Tabela A.64: IOPS para escritas aleatórias num volume RAID-10, configurado com quatro discos, dois dos quais como discos de paridade.

Apoios

Este trabalho foi financiado pelo Fundo Europeu de Desenvolvimento Regional (FEDER) através do Programa Operacional Temático Factores de Competitividade (COMPETE) e por Fundos Nacionais através da Fundação para a Ciência e Tecnologia (FCT), no âmbito dos projetos "Suporte para o *tier-2* de ATLAS e CMS no contexto do memorando de entendimento do WLCG"(RECI/FIS-NUC/0115/2012), e "Piloto de Experimentação de *Cloud Computing*".